

精通 Oracle 12c 数据库管理

Mastering Oracle Database Management

- 面向初学者的Oracle培训书，方便快速入门
- 掌握Oracle数据库核心技术，胜任企业大型数据库管理、维护和开发工作

王荣鑫 著

清华大学出版社



精通 Oracle 12c 数据库管理

王荣鑫 著



清華大學出版社
北京

内 容 简 介

本书以引导读者快速实践 Oracle 12c 数据库为原则，由浅入深，涵盖 Oracle 12c 数据库管理的主要实践活动，内容非常贴合实际管理需要。

本书共 12 章。第 1~4 章介绍 Oracle 数据库基础知识，如 Oracle 的发展史、数据库简单的基本安装、数据库体系结构、数据库自动存储管理等。第 5～8 章介绍数据库日常运维的基本工作内容，主要有数据库的备份和恢复、数据库优化以及常用的数据库运维工具使用方法。第 9 章介绍 Oracle 12c 新特性的多租户功能。第 10~11 介绍 SQL 语句和 PL/SQL 的使用方法。第 12 章是 RAC 案例部分，介绍 Oracle 数据库的高级安装方法。

本书内容丰富，范例精典，实用性强，适合初级、中级层次想要精通 Oracle 数据库技术的人员阅读，尤其适合数据库进阶人员或高校相关专业的师生阅读。

图书在版编目（CIP）数据

精通 Oracle 12c 数据库管理/王荣鑫著. — 北京：清华大学出版社，2018
（数据库技术丛书）
ISBN 978-7-302-51357-5

I. ①精… II. ①王… III. ①关系数据库系统 IV. ①TP311.138

中国版本图书馆 CIP 数据核字（2018）第 229163 号

责任编辑：夏毓彦
封面设计：王　翔
责任校对：闫秀华
责任印制：

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编：**100084
社 总 机：010-62770175　　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn

印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：190mm×260mm　　**印　　张：**18.25　　**字　　数：**467 千字
版　　次：2018 年 12 月第 1 版　　**印　　次：**2018 年 12 月第 1 次印刷
定　　价：59.00 元

产品编号：074007-01

前　言

Oracle 将只会做一件事情，我们管理海量的数据并通过网络提供这些数据。

——Oracle 创始人 Larry Ellison

作为一名 IT 领域的工作者，无论是偏重于程序的开发，还是基础平台的运维，都不可避免地要与数据库打交道。目前市场中，由于 Oracle 体系结构的灵活与复杂性，导致很多读者认为学习 Oracle 是一件非常困难的事情。其实不然，学习 Oracle 数据库可以作为一个入门的功课，对未来了解不同种类的数据库软件打下良好的基础。为了让人们更容易学习 Oracle，本书以初学者为对象，用简洁的语言讲解 Oracle 数据库系统。

Oracle 数据库系统是 Oracle 公司推出的跨平台的具有高稳定性、高可用性和灵活体系结构的数据库管理软件，可以在 Linux、Windows、UNIX 等系统上运行，并且具有一致的操作方式。目前市面上很多优秀的 Oracle 书籍都是以某个特定的技术点为基点，或者针对某些特定的案例，或侧重于开发，或侧重于管理，使学习 Oracle 的读者不能全局领略 Oracle 的组成和日常的基本工作内容。本书作者以初学者为对象，使用最简单易懂的语言，力求让读者轻松学习 Oracle，使读者能深入理解 Oracle 数据库的体系结构，了解作为一名数据库管理员日常的基本工作内容和常用的技能。本书各个章节中列举了大量具体实例，以强化读者对于 Oracle 知识点的理解。

本书特色

1. 循序渐进，由浅入深

为了能使读者更好地理解 Oracle 数据库的基础知识，本书首先介绍了数据库的基本架构，并为读者讲解了安装 Oracle 数据库系统的过程，通过层层深入的方式带领读者进入 Oracle 的世界。

2. 内容充实，层次清楚

本书层层深入，从 Oracle 系统的基本操作开始，详细讲解了数据库对象的创建与管理，并详细介绍了 PL/SQL 的编程知识，通过理论与实例结合的方式让读者尽快对 Oracle 的管理与开发有一个清楚的认识。

3. 实例丰富，步骤清晰

本书的每个知识点都通过简单易懂的例子进行剖析，训练读者的操作思路，使初学者少走弯路。

4. 案例指导，提供脚本

本书在每章讲解之后，针对本章要点进行总结，通过案例巩固知识要点和操作要点，提供详细的源码脚本。

本书内容体系

本书分为 12 章，内容介绍如下。

第 1～4 章主要讲解 Oracle 数据库基础知识，包括 Oracle 的体系结构、简单的基础安装和数据库逻辑层面的基础知识。

第 5～8 章主要讲解 Oracle 数据库日常运维内容，作为一名数据库管理员，这部分内容是首先需要了解和掌握的，尤其是数据库优化的部分，实践中会有大量的场景需要接触。

第 9 章主要讲解 Oracle 12c 新特性，包括 Oracle12c 的多租户功能。多租户功能作为 Oracle 的一项重大变化，是需要数据库管理员重点掌握的知识点。

第 10～11 章主要讲解 Oracle 数据库开发技能，包括 SQL 语句和 PL/SQL 的相关知识，以举例的形式方便读者理解代码的功能和编写技巧。

第 12 章是一个 RAC 的案例实战，主要介绍基于 Oracle 12c 版本的高级安装。RAC 架构是企业级的常用架构，掌握该案例的安装方法对实际项目和日常数据库管理工作有重要的作用。

适合阅读本书的读者

- Oracle 数据库管理初学者
- Oracle 程序设计人员
- 系统开发人员
- 高等院校计算机相关专业师生
- 数据库培训人员
- 数据库管理人员（DBA）

本书由王荣鑫主笔，其他参与创作的还有王晓华、刘鑫、陈素清、林龙、王亚飞、薛燚、王刚、李雷霆、李一鸣、谢志强、王启明、罗从良，排名不分先后。

王荣鑫

2018 年 10 月

目 录

第 1 章

Oracle 12c数据库软件安装

Oracle 数据库软件作为目前大型企业，尤其是金融行业最流行的关系型数据库，一直深受用户的认可，主要是因为其在关系型数据库领域有很好的性能。直到今天，Oracle 数据库在各大数据库排行榜中也是独占鳌头。

比较流行的数据库还包括 MySQL。它是一个开源的产品，现在被 Oracle 公司收购。当然作为一款开源产品，其最主要的优势是省钱，但同时也给使用者带来了比较大的压力，因为如果想用好这款产品，就必须对它有深入的了解，同时具有二次开发的强悍实力，相比之下，Oracle 数据库软件就显得友好很多。

Oracle 数据库有着非常多的个人爱好者，甚至可以说，Oracle 数据库的爱好者群体是所有数据库里面最多的也不为过。在网上，你可以搜到大量的相关学习资料，全部是免费公开的。并且，Oracle 数据库的所有软件都可以免费下载，免费体验学习，我们只需要承诺不用作商业用途即可。所以，Oracle 数据库既是一款付费的商业软件，又是一款非常开放的软件。如果想学习 Oracle 数据库，除了教材之外，也可以通过搜索来了解 Oracle 数据库的更多功能和组件。Oracle 数据库的入门相对来讲是比较容易和轻松的。

现在 Oracle 数据库历经多年的发展，已经演变到 12c 版本，从版本号可以看出，Oracle 数据库的发展经过了比较长时间的过程：

- 1977 年 6 月，Larry Ellison 与 Bob Miner 和 Ed Oates 在硅谷共同创办了一家名为软件开发实验室（Software Development Laboratories，SDL）的计算机公司（Oracle 公司的前身）。
- 1979 年，SDL 更名为关系软件有限公司（Relational Software，Inc.，RSI）。
- 1983 年，为了突出公司的核心产品，RSI 再次更名为 Oracle。Oracle 从此正式走入人们的视野。到 1983 年这个过程中，Oracle 其实已经产生了 3 个早期的版本。
- 1988 年，Oracle 发布了 6.0 版，之后 Oracle 不断完善。
- 在 2001 年 6 月的 Oracle Open World 大会中，Oracle 发布了 Oracle 9i。在 Oracle 9i 的诸多新特性中，最重要的就是 Real Application Clusters（RAC）了，也就是 Oracle 的集群特性，该特性一直影响 Oracle 至今，在很多企业中，RAC 已经是必备的高可用架构。
- 对于现在的使用者来说，9i 也是一个很古老的版本。之后，2003 年推出了 10g。

- 2007 年推出的 12c，是目前市场上广泛使用的版本，尤其是 12cR2 版本。

本书以 Oracle 12c 版本作为讲解对象，将各位读者带入 Oracle 数据库世界。

1.1 Oracle 数据库安装

在计算机软件学习中，有一种方法叫“从安装开始”。除了本书介绍的 Oracle 数据库软件之外，在日常工作生活中，凡是遇到一个新的软件，大家都有一个习惯，就是不管是否了解该软件，先安装到电脑上试试，以尝试的心态开始学习之路。之后的学习，包括了解其运行的原理、架构、使用方法等，都是在安装的基础之上。Oracle 数据库软件也是如此，你必须有一套环境，没有数据库环境，后面的学习内容就无从谈起。

本节主要讲解 Oracle 数据库软件的安装，包括基础环境的准备和安装过程。

1.1.1 操作系统镜像文件下载

安装 Oracle 数据库之前，需要准备操作系统环境，本书选择 Oracle Linux 作为系统环境，使用版本为 Oracle Linux 6。

Oracle Linux 的全称为 Oracle Enterprise Linux，简称 OEL，是 Oracle 公司在 2006 年初发布的第一个版本，是 Linux 发行版本之一，以对 Oracle 软件和硬件支持较好见长。OEL 一般被称为 Oracle 企业版 Linux，由于 Oracle 提供的企业级支持计划 UBL（Unbreakable Linux），因此很多人都称 OEL 为坚不可摧 Linux。2010 年 9 月，Oracle Enterprise Linux 发布新版内核——Unbreakable Enterprise Kernel，专门针对 Oracle 软件与硬件进行优化，最重要的是 Oracle 数据库跑在 OEL 上性能可以提升超过 75%。

Oracle Enterprise Linux 是由 Oracle 公司提供支持的企业级 Linux 发行。据项目网站称，Oracle 以 Red Hat Linux 作为起始，移除了 Red Hat 的商标，然后加入了 Linux 的错误修正。Oracle Enterprise Linux 旨在保持与 Red Hat Enterprise Linux 完全兼容。

Oracle Enterprise Linux 与 Red Hat Enterprise Linux 二进制兼容，也就是能运行在 Red Hat 上的软件也能运行在 Oracle 上。Oracle 的想法是加强 Linux，而不是从零开始创造一个，Oracle 发行版实际上有与 Red Hat 完全一样的血统，它们是同父同母的兄弟，Oracle Linux 与 Red Hat 这个兄弟最大的区别就是 OCFS（Oracle Cluster File System）文件系统和基于 Xen 的 OracleVM 虚拟化技术。Oracle Enterprise Linux 目前支持 x86 和 x86_64 两种平台。

Oracle 公司表示，Oracle Enterprise Linux 不是人们所认为的复制 Red Hat，最重要的是向用户提供高品质的操作系统，并提供充分的支持，为在 Oracle Linux 上运行的整个应用程序体系提供全面的专业服务，包括数据库、中间件、应用程序、管理工具和操作系统本身的全面服务。

Oracle Enterprise Linux 的下载地址为 https://edelivery.Oracle.com。Oracle 公司的软件产品多数都可以在该网站获得下载，其为 Oracle Software Delivery Cloud，即 Oracle 软件交付云平

台。本书所使用的 Oracle 数据库软件同样可以在该网站下载。从实际的使用情况看，相关产品软件的早期版本或测试版都会首先在此平台进行发布，发布时间往往略早于 Oracle 官网的首页（见图 1-1）。

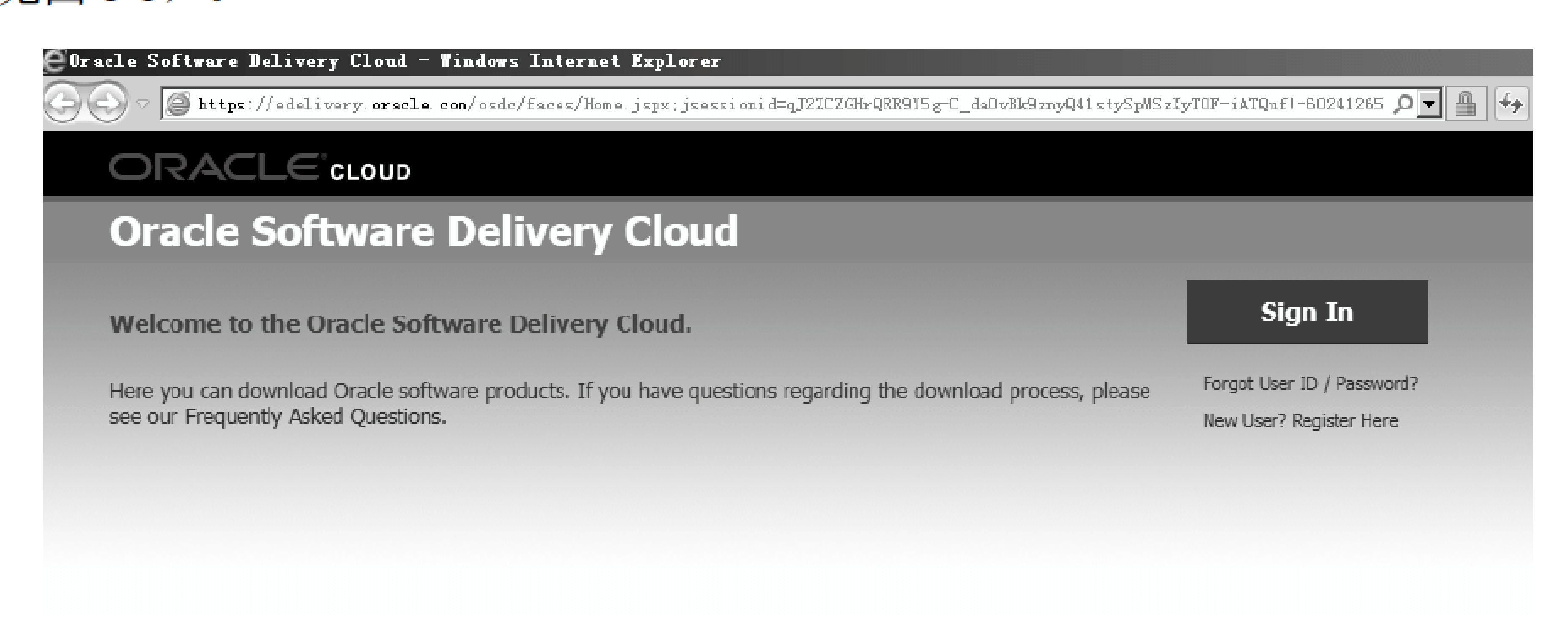


图 1-1　Oracle 官网网站的首页

（1）登录网站所需的用户名和密码同登录 Oracle 官网一致，这方面的安全策略为使用者提供了便利。申请一个 Oracle 账户就可以在不同的站点之间登录，包括 Oracle 的官网网站、Oracle 的服务支持站点、软件交付云平台和社区网站等。接着读者可以进行 Oracle 账户的创建（创建过程略）。请根据提示填写基本信息并注册，注册成功之后，使用刚刚注册的用户名和密码登录网站，如图 1-2 所示。

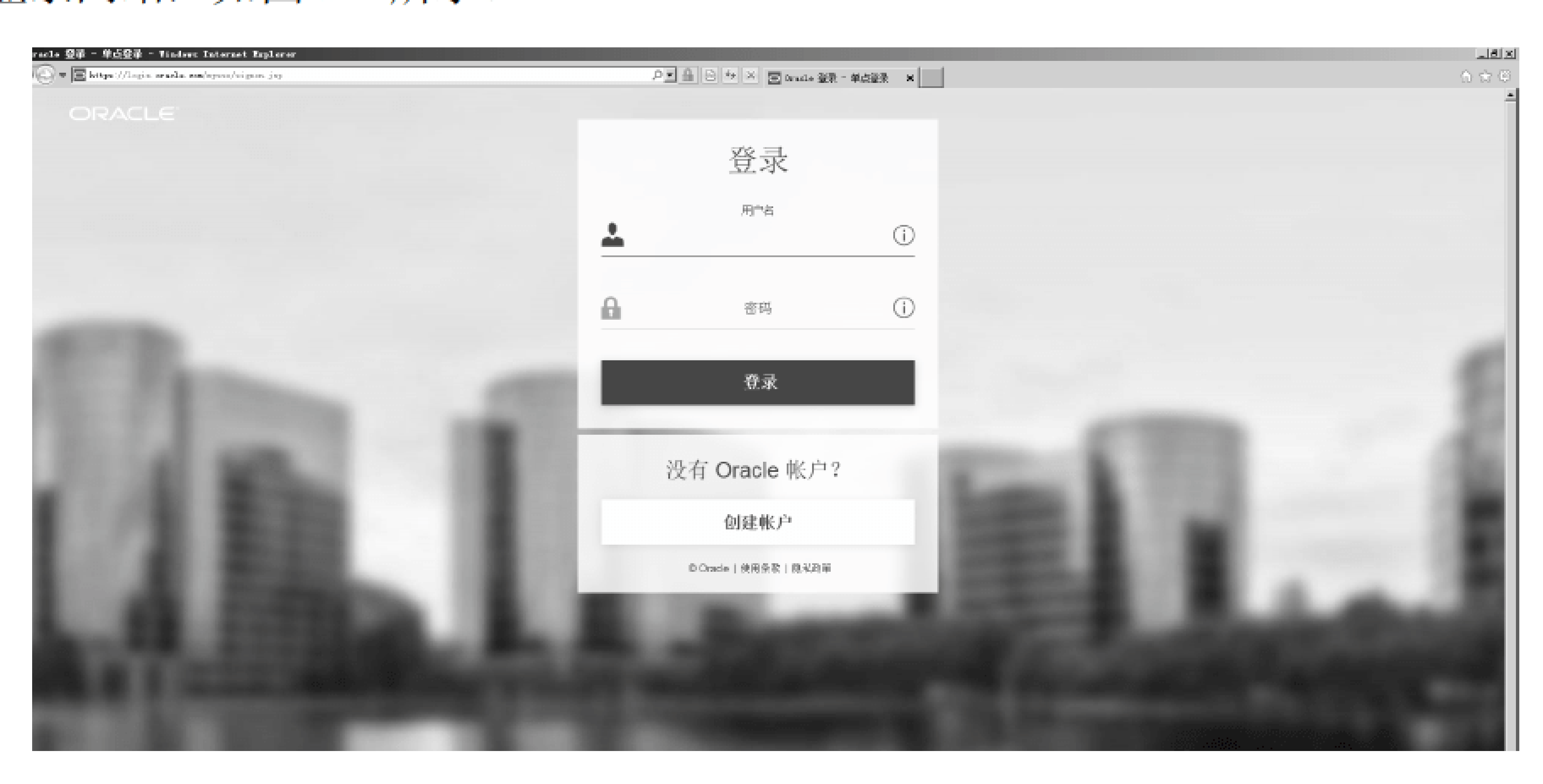


图 1-2　登录页面

（2）进入该网站之后，可以看到如图 1-3 所示的软件选择功能页面，根据需要选择要下载的产品。对于 Filter Products 选项，一般来说使用默认选择即可，不需要进行修改。值得注意的是，Oracle 绝大部分的产品都是兼容多个平台的，不同平台有不同的安装包，请根据实际情况选择正确的平台，在 Select platform 选项修改。

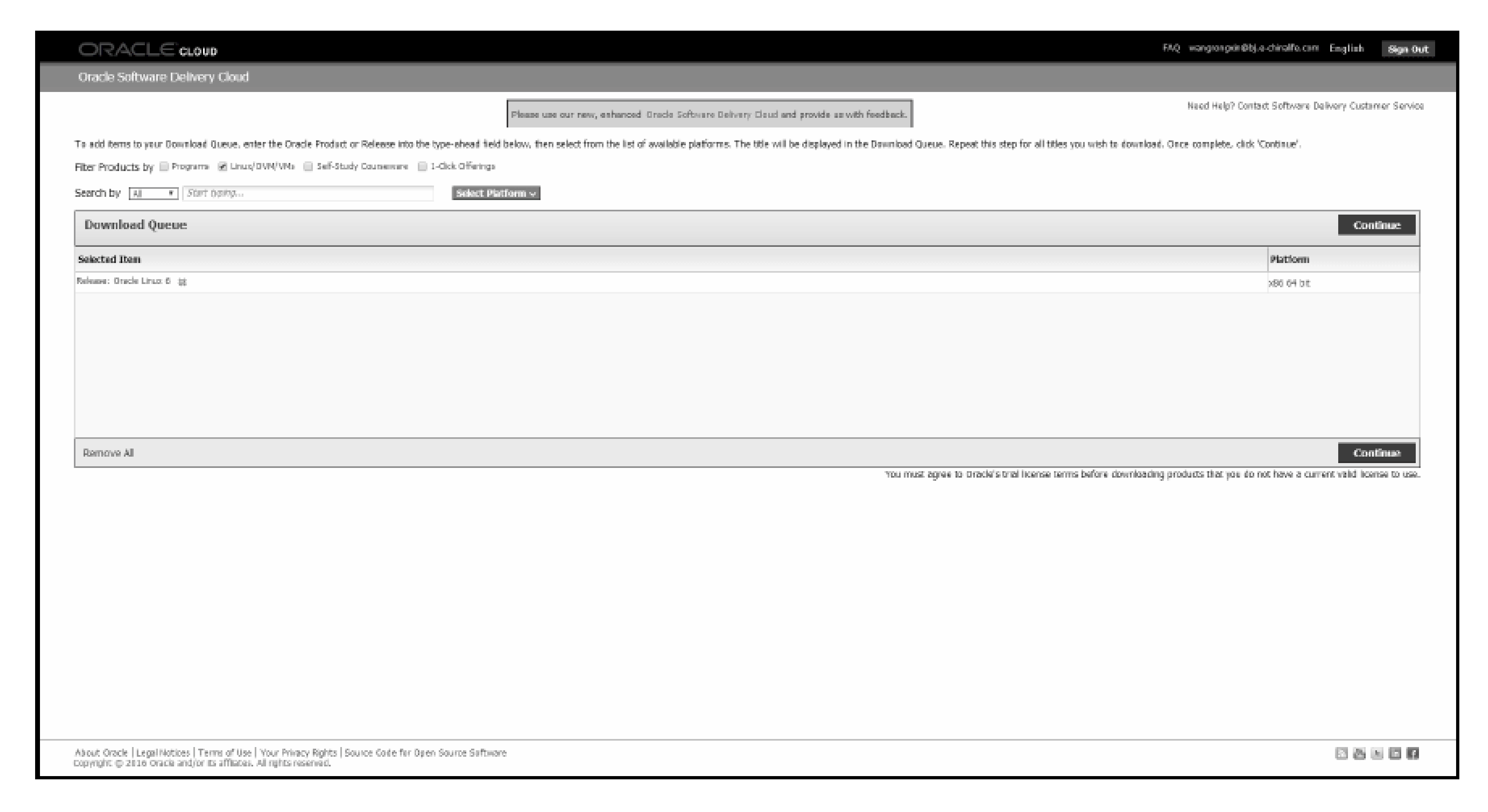


图 1-3　软件选择功能页面

（3）本书用例环境使用的操作系统环境为 Oracle Linux 6 版本，只能在 Oracle 软件交付云平台下载。Oracle 官方网站没有具体的下载链接，并且会提醒用户需要跳转至该平台。可使用筛选功能定位安装介质：

- Search by 选择 release。
- 产品选择 Oracle Linux 6。
- Select platform 选择 x86 64 bit。

（4）筛选器选择完毕之后，进入软件平台选择页面，如图 1-4 所示。在下面的介质列表中会显示已经选定的软件信息，之后单击 Continue 按钮进入下一页面。

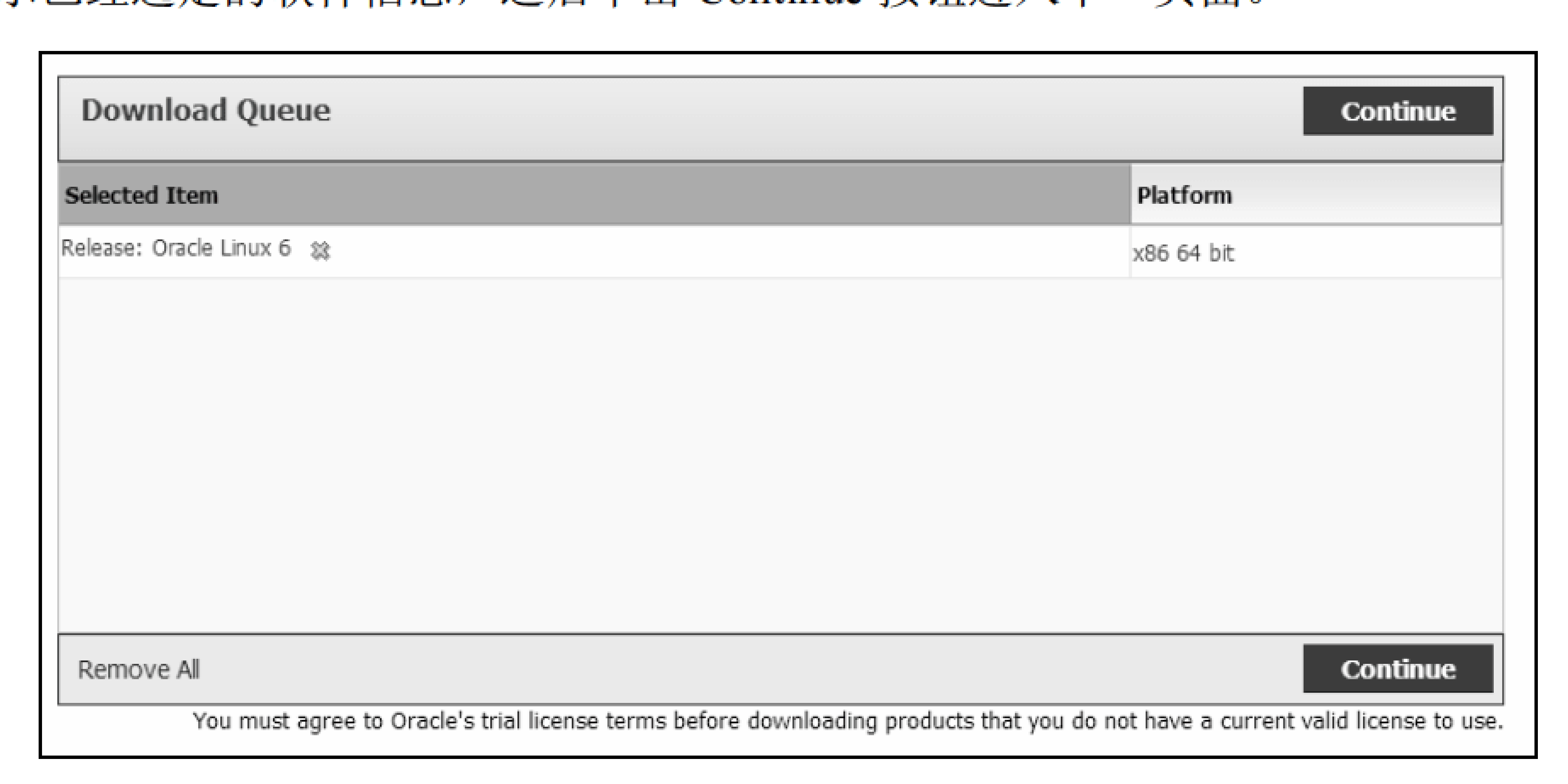


图 1-4　软件平台选择页面

（5）在此页面系统会继续将所选软件的详细信息罗列出来，以便使用者能更好地判断正确与否，包括软件版本、软件名称、安装包大小，以及最后的更新时间。确认无误之后可以继续单击 Continue 按钮进入下一页面，如图 1-5 所示。

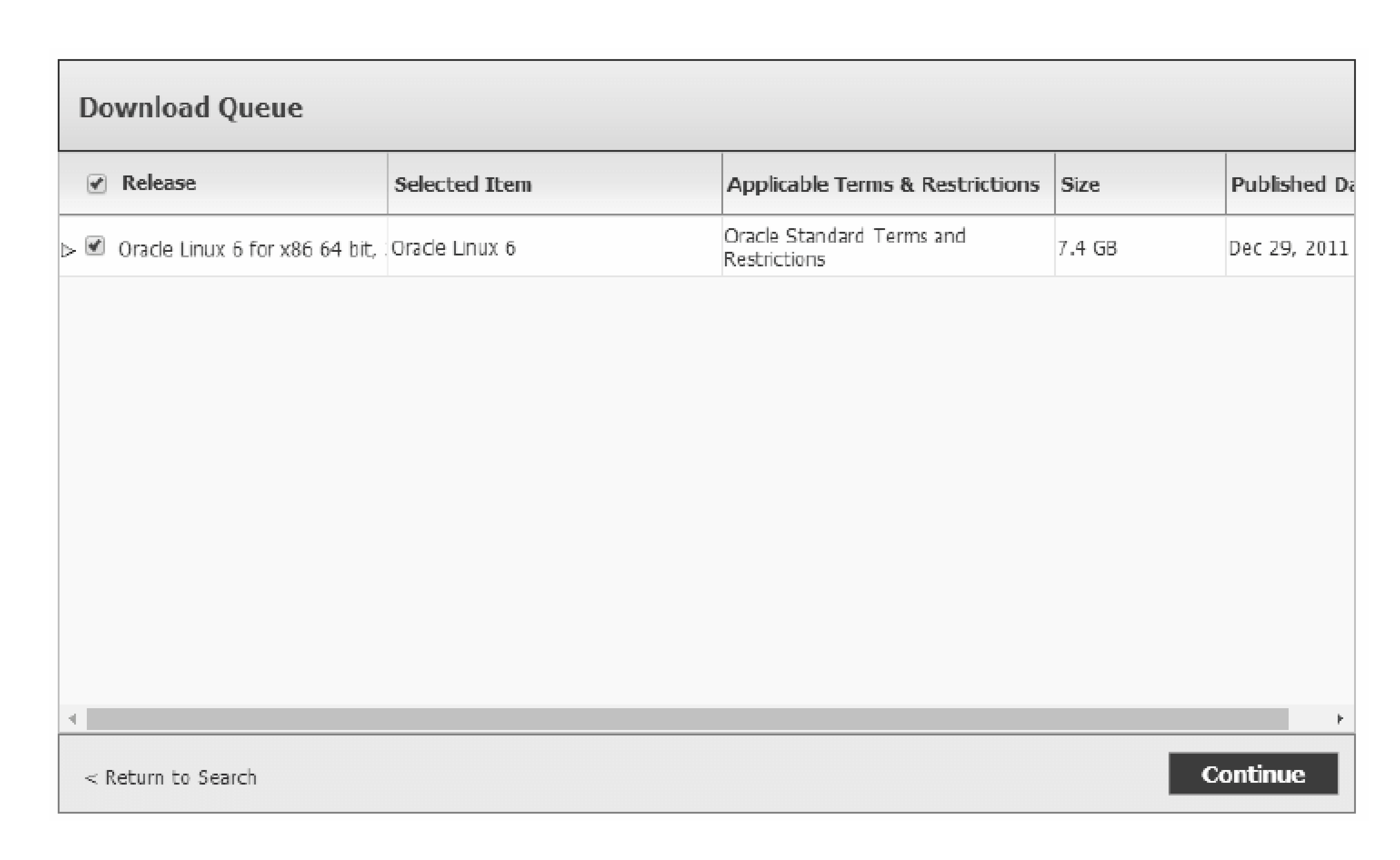


图 1-5　软件详细信息

（6）之后系统会将软件的简介、版权协议等信息展现出来，需要使用者进行确认，并同意以上说明。如果需要下载必须同意该协议，一般来说都会选择同意。Oracle 软件的一大特点就是安装时不需要使用 License 激活。当软件的使用者以测试或学习为目的时，可以在 Oracle 官方免费下载，并且可以免费查阅大量详细的软件使用手册。这种策略有利于 Oracle 软件的推广。确认同意之后继续单击 Continue 按钮进入下一页面，如图 1-6 所示。

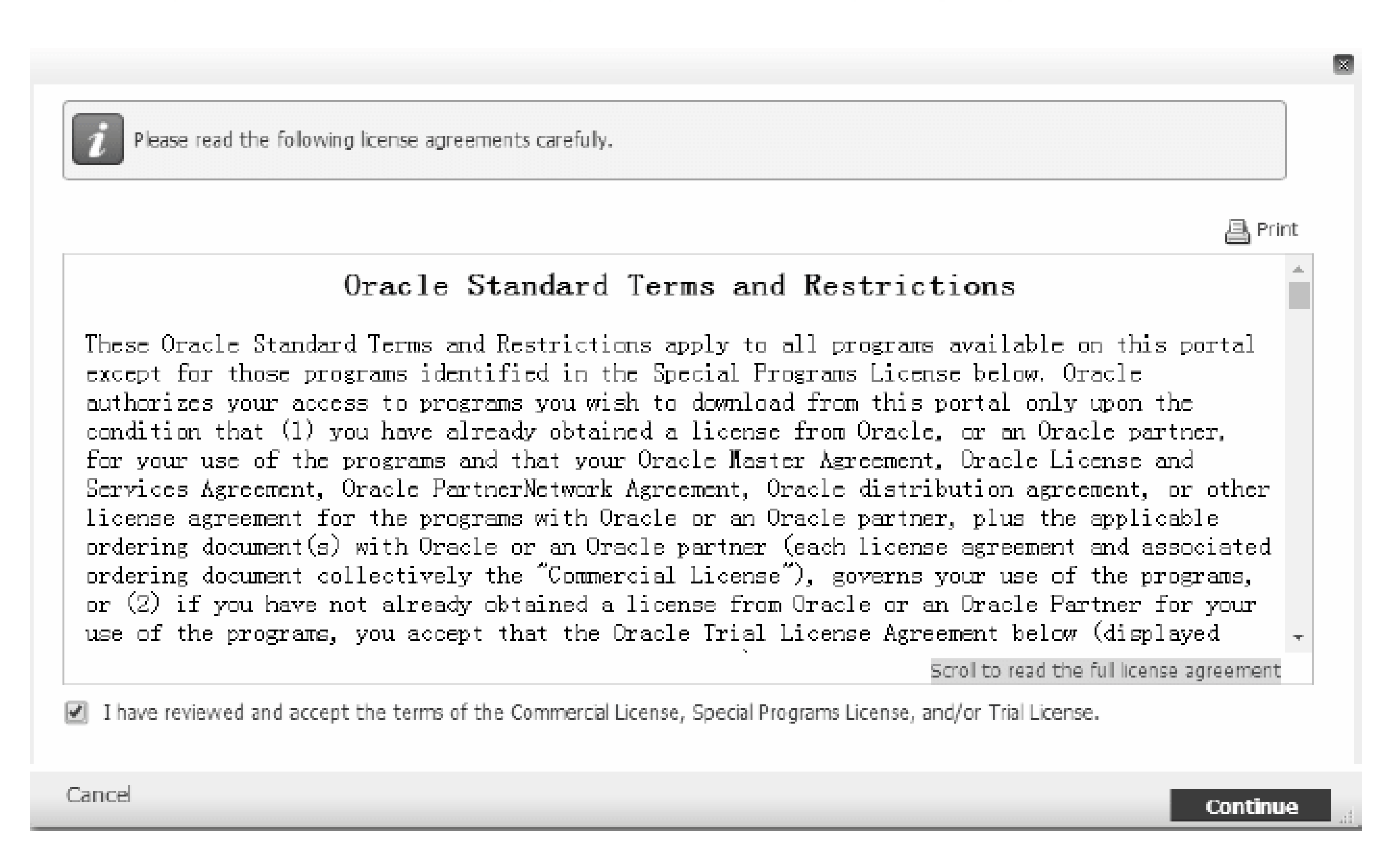


图 1-6　软件下载协议

（7）Oracle Linux 系统安装介质的镜像文件分两种：单一安装文件和分片安装文件。两种安装介质没有较大的区别，可以选在单一文件进行下载。选定之后单击 Download 按钮下载，也可以直接单击软件包名称，单击之后即开始下载，如图 1-7 所示。

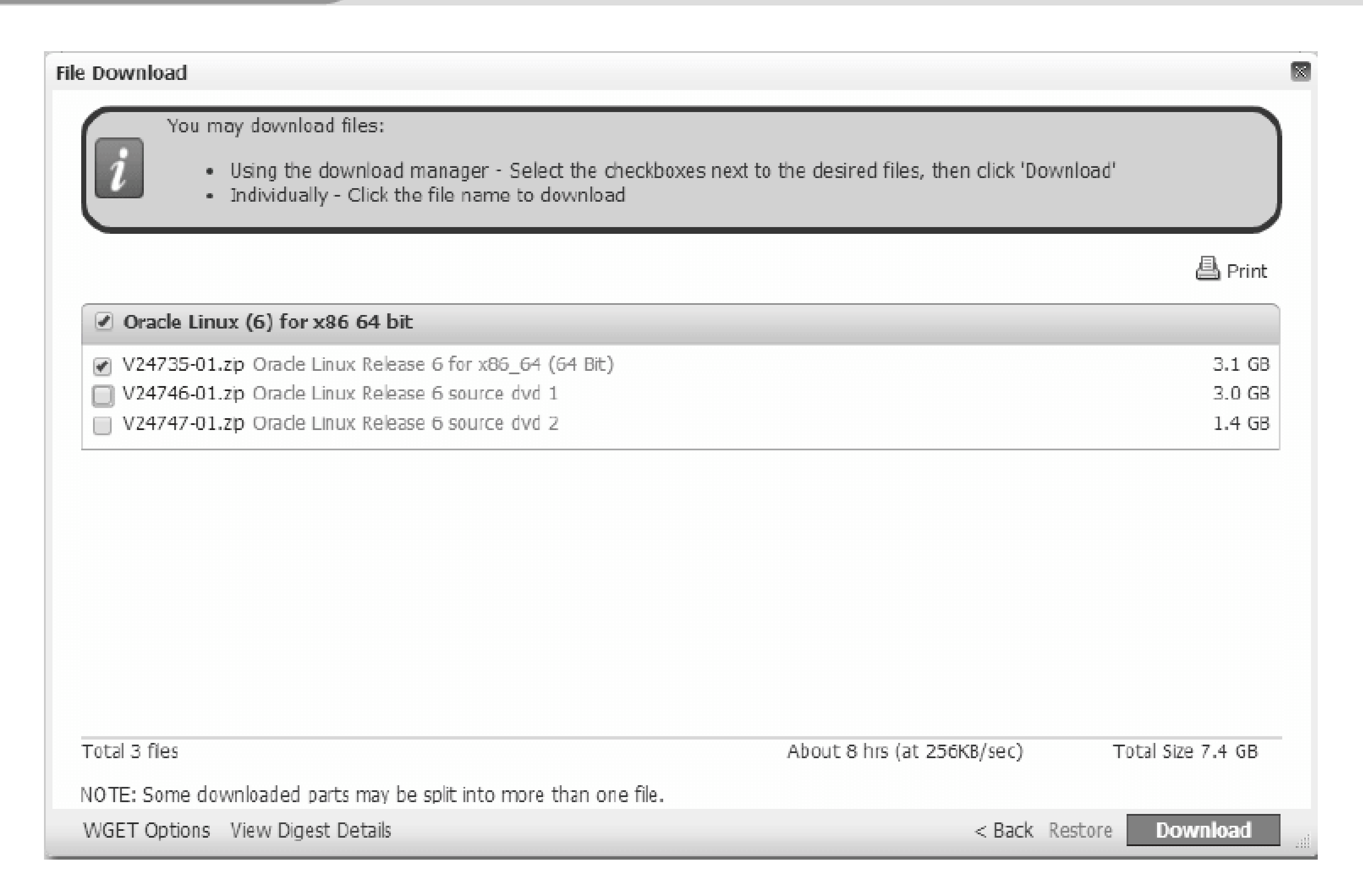


图 1-7　软件压缩包列表

1.1.2　操作系统配置方法

由于数据库需要指定的软件包、软件包版本以及内核参数调整，因此当在系统上安装 Oracle Database 12c 之前，需要预先配置操作环境。

在 Oracle Linux 上，有一种非常轻松的办法可以让系统满足这些安装先决条件：首先安装一个名为 Oracle-Database-Server-12cR2-preinstall 的 RPM 软件包。此 RPM 执行一些预配置步骤，包括：

（1）自动下载并安装 Oracle Grid Infrastructure 和 Oracle Database 12c 第 2 版（11.2.0.3）所需的任何额外软件包和特定软件版本，并通过 yum 或 up2date 功能处理软件包依赖关系。

（2）创建用户 oracle 和组 oinstall（针对 OraInventory）、dba（针对 OSDBA），供数据库安装期间使用（出于安全目的，该用户没有默认口令，且不能远程登录）。要启用远程登录，请使用 passwd 工具设置一个口令。

（3）修改/etc/sysctl.conf 中的内核参数以更改共享内存、信号、最大文件描述符数量等设置。

（4）设置/etc/security/limits.conf 中的软硬 Shell 资源限制，如锁定内存地址空间、打开的文件数量、进程数和核心文件大小。

（5）对于 x86_64 计算机，在内核中设置 numa=off。



Oracle-Database-Server-12cR2-preinstall 只是根据数据库安装的需要来分析现有的/etc/sysctl.conf 和/etc/security/limits.conf 文件并更新值。所有与数据库安装无关的预自定义设置保持不变。

Oracle-Database-Server-12cR2-preinstall RPM 软件包可通过 Oracle Unbreakable Linux Network（ULN，需要支持合同）、Oracle Linux 分发媒体或 Oracle 公共 yum 信息库获取。因

此，无论系统是否在 ULN 注册访问 Oracle 补丁和支持，均可以使用 Oracle-Database-Server-12cR2-preinstall 来简化 Oracle Linux 上的数据库安装。此外，Oracle 公共 yum 信息库现在还包括了所有安全和错误勘误表，从而通过最新的安全更新和错误修复来确保系统的安全和稳定。

以下是针对 Oracle 数据库安装使用 Oracle-Database-Server-12cR2-preinstall 对系统进行预配置的步骤。

作为一个授权用户（如 root），检索配置信息库位置的文件。

【示例 1-1】检索配置信息库位置的文件

```
# cd /etc/yum.repos.d
# wget http://public-yum.Oracle.com/public-yum-ol6.repo
```

使用文本编辑器修改该文件，将字段 enabled=0 更改为 enabled=1，以反映对应于该计算机操作系统版本的信息库。

【示例 1-2】public-yum-old6.repo 的部分内容

```
[ol6_latest]
name=Oracle Linux $releasever Latest ($basearch)
baseurl=http://public-yum.Oracle.com/repo/OracleLinux/OL6/latest/$basearch/
gpgkey=http://public-yum.Oracle.com/RPM-GPG-KEY-Oracle-ol6
gpgcheck=1
enabled=1
[ol6_UEK_latest]
name=Latest Unbreakable Enterprise Kernel for Oracle Linux $releasever ($basearch)
baseurl=http://public-yum.Oracle.com/repo/OracleLinux/OL6/UEK/latest/$basearch
/
gpgkey=http://public-yum.Oracle.com/RPM-GPG-KEY-Oracle-ol6
gpgcheck=1
enabled=1
```

因为目标系统运行的是适用于 x86_64 的 Oracle Linux 第 6 版 Update 6，所以要启用[ol6_latest]和[ol6_UEK_latest]信息库。接下来，使用 yum install 命令安装 Oracle-Database-Server-12cR2-preinstall RPM。清单中的输出显示了安装过程如何检查依赖关系，然后下载和安装所需软件包。

【示例 1-3】使用 yum install 命令安装 Oracle-Database-Server-12cR2-preinstall RPM

```
# yum install Oracle-Database-Server-12cR2-preinstall
Loaded plugins:refresh-packagekit, rhnplugin, security
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package Oracle-Database-Server-12cR2-preinstall.x86_64 0:1.0-6.el6 will be
installed
……
--> Finished Dependency Resolution
Dependencies Resolved
```

```
================================================================================
Package                     Arch   Version             Repository  Size
================================================================================
Installing:
Oracle-Database-Server-12cR2-preinstall
x86_64 1.0-6.el6            ol6_latest  15 k
Installing for dependencies:
cloog-ppl                    x86_64 0.15.7-1.2.el6       ol6_latest  93 k
compat-libcap1                x86_64 1.10-1              ol6_latest  17 k
compat-libstdc++-33            x86_64 3.2.3-69.el6        ol6_latest 183 k
cpp                         x86_64 4.4.6-4.el6          ol6_latest 3.7 M
gcc                         x86_64 4.4.6-4.el6          ol6_latest  10 M
gcc-c++                      x86_64 4.4.6-4.el6          ol6_latest 4.7 M
glibc-devel                   x86_64 2.12-1.80.el6_3.4     ol6_latest 970 k
glibc-headers                 x86_64 2.12-1.80.el6_3.4     ol6_latest 600 k
kernel-uek-headers             x86_64 2.6.32-300.32.1.el6uek ol6_latest 713 k
ksh                         x86_64 20100621-16.el6       ol6_latest 684 k
libaio-devel                  x86_64 0.3.107-10.el6        ol6_latest  13 k
libstdc++-devel                x86_64 4.4.6-4.el6          ol6_latest 1.5 M
mpfr                        x86_64 2.4.1-6.el6          ol6_latest 156 k
ppl                         x86_64 0.10.2-11.el6        ol6_latest 1.3 M
Transaction Summary
================================================================================
Install     15 Package(s)
Total Download size:25 M
Installed size:61 M
Is this ok [y/N]:Downloading Packages:
------------------------------------------------------------------------------
--
Total                                       710 kB/s |  25 MB    00:35
Running rpm_check_debug
Running Transaction Test
Transaction Test Succeeded
Running Transaction
Installing :mpfr-2.4.1-6.el6.x86_64                               1/15
……
mpfr.x86_64 0:2.4.1-6.el6
ppl.x86_64 0:0.10.2-11.el6
Complete!
```

yum 安装过程在/var/log/Oracle-Database-Server-12cR2-preinstall/results/orakernel.log 文件中记录有关内核更改的消息，并在/var/log/Oracle-Database-Server-12cR2-preinstall/backup 目录中备份当前系统设置。

至此，系统已准备好，可以安装 Oracle 数据库了。



本文适用于 Oracle Linux 6，针对 Oracle Linux 的 Oracle-Database-Server-12cR2-preinstall RPM 简介。

1.1.3　Oracle 数据库软件安装程序下载方法

Oracle 数据库软件在不同的操作系统平台的运行机制有所不同，主要区别是 UNIX/Linux 与 Windows 系统上面的区别。并且，Oracle 数据库软件的安装介质（安装程序）对应不同的操作系统，读者可以在 Oracle 的官方网站上进行下载，下载地址为 http://www.Oracle.com/technetwork/Database/enterprise-edition/Downloads/index.html。

（1）进入下载页面，在正式下载之前，请选中 Accept License Agreement 单选框，确认许可授权方可开始下载。本书采用 Linux 操作系统作为实例演示环境，选择 Linux x86-64 版本，单击 See All 按钮，进入下一页面，具体方法如图 1-8 所示。

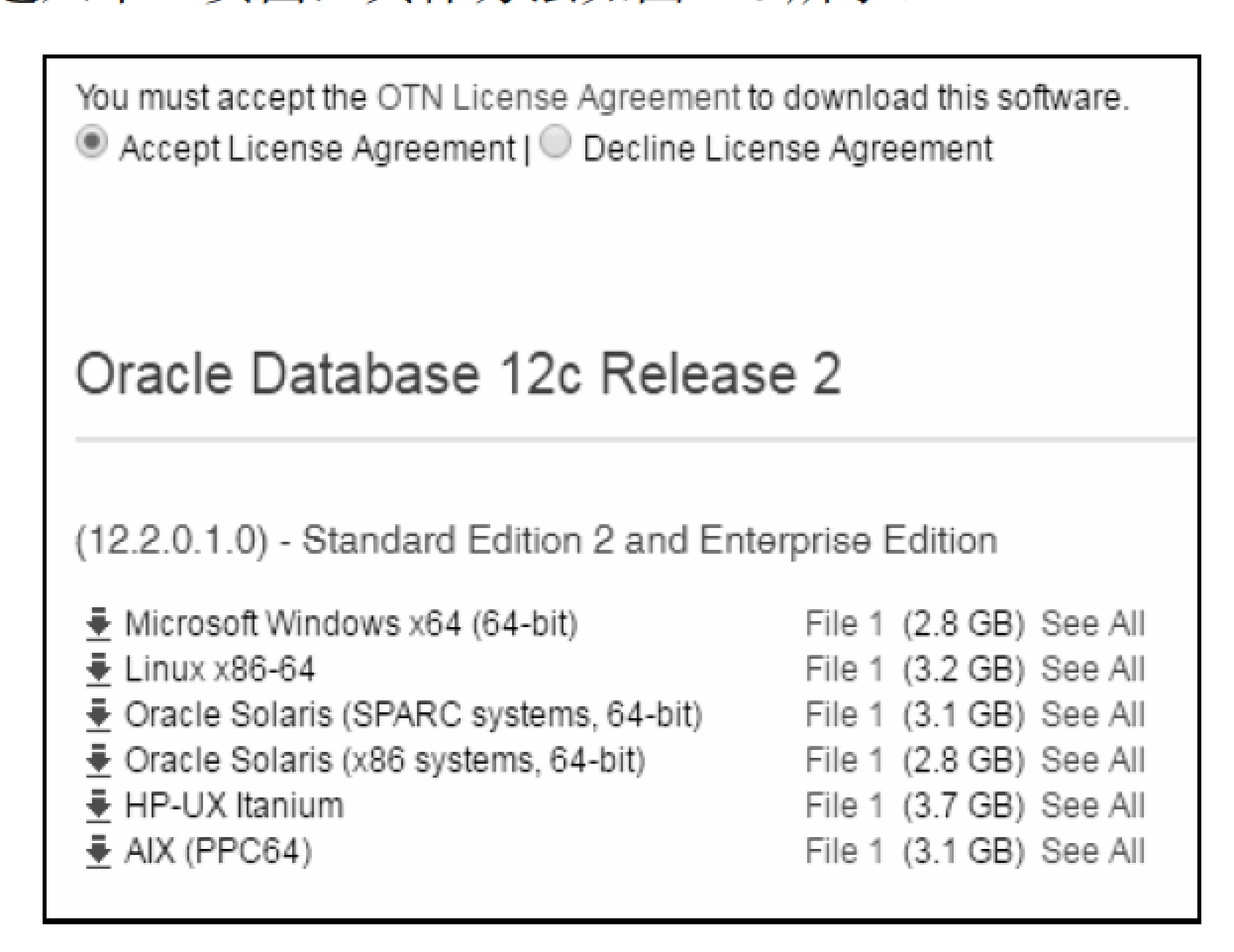


图 1-8　Oracle 数据库软件版本选择

（2）对于企业来说，会下载 Linux 或 UNIX 版本的程序包，个人学习时，可以将电脑当作一台数据库服务器，下载安装 Windows 版本的 Vmware Workstation 软件，创建测试使用的虚拟机环境，并且虚拟机安装 Linux 操作系统。本书的实验练习环境采用 Linux 平台，如果读者的电脑是苹果或者其他版本的 Linux 系统，请下载对应正确的安装包。其实无论是什么版本，Oracle 软件安装之后的使用都是基本相同的，并无大的差别。

（3）本书采用 Linux 操作系统作为实例演示环境，故下载对应的 Linux x86-64 软件安装包。请单击 linuxx64_12201_database.zip 链接进行下载，如图 1-9 所示。

Oracle Database 12c Release 2 (12.2.0.1.0) for Linux x86-64
linuxx64_12201_database.zip (3,453,696,911 bytes) (cksum - 4170261901)

Directions
1. All files are in the .zip format. There is an unzip utility here if you need one.
2. Download and unzip both files to the same directory.
3. Installation guides and general Oracle Database 12c documentation are here.

图 1-9　Oracle 数据库软件 Linux 版本下载链接

1.1.4 基础环境配置检查

随着 IT 技术的快速发展，当前绝大部分的个人或企业用 PC Server 的硬件配置都得到了很高的提升，尤其是 CPU 和内存技术的更新迭代迅速，使用者基本不需要再为 Oracle 数据库软件对硬件的要求发愁，目前几乎所有的主机都满足 Oracle 数据库软件的硬件最低要求。本书的演示实例环境使用 Linux 操作系统，配置如表 1-1 所示。

表 1-1　Oracle 数据库软件安装环境配置要求

检查项	配置要求
服务器架构	确认服务器构成、模型、核心架构和主机总线适配器（HBA）或网络接口控制器（NIC）都支持运行 Oracle 数据库和 Oracle 网格基础设施
最低内存	2GB
视频适配器	256 色
显示器分辨率	最低 1024×768

基础环境确认准备完毕之后就可以启动正式的安装程序了。

1.1.5 运行安装程序

解压下载的数据库安装软件包压缩文件。进入解压之后的文件夹，会显示如图 1-10 所示的数据库软件安装文件列表。

```
drwxr-xr-x  2 oracle oinstall  4096 Aug 27  2013 sshsetup
-rwxr-xr-x  1 oracle oinstall  3267 Aug 27  2013 runInstaller
drwxr-xr-x  2 oracle oinstall  4096 Aug 27  2013 rpm
drwxr-xr-x  2 oracle oinstall  4096 Aug 27  2013 response
-rw-r--r--  1 oracle oinstall 30016 Aug 27  2013 readme.html
drwxr-xr-x 14 oracle oinstall  4096 Aug 27  2013 stage
-rw-r--r--  1 oracle oinstall   500 Aug 27  2013 welcome.html
drwxr-xr-x  4 oracle oinstall  4096 Aug 27  2013 install
```

图 1-10　数据库软件安装文件

执行./runInstaller，启动数据库软件安装程序，软件会加载并初步校验系统是否可以达到数据库安装的最低配置。如果达到要求，就会直接加载程序并进行下一步的安装，相对于 UNIX/Linux 操作系统，Windows 系统平台上安装 Oracle 数据库软件简单了很多，省去了修改各种操作系统参数。

1.1.6 配置安全更新

（1）配置安全更新界面如图 1-11 所示。该界面提示是否希望通过 Oracle support 接收安全更新，一般来说是不勾选，因为在绝大部分的企业内部，Oracle 数据库软件的更新补丁、版本升级都是需要人工控制的，况且该选项需要主机与 Oracle support 之间有互联网联通。

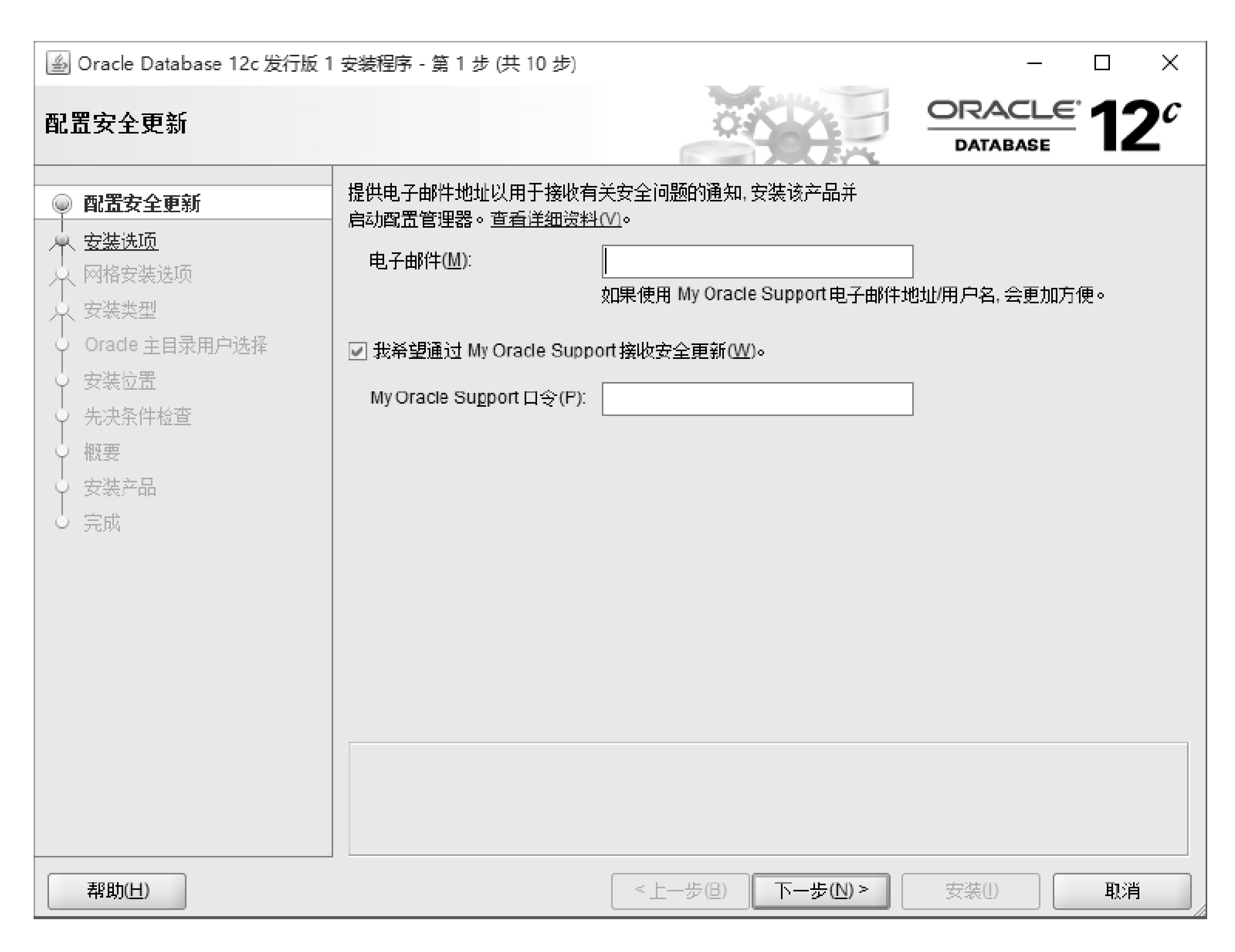


图 1-11　确认是否需要自动接收安全更新

（2）确认不希望通过 Oracle support 接收安全更新。此处选项可以跳过：不填写电子邮件项，取消下面的复选框勾选，不要求通过 Oracle support 进行软件安全更新，如图 1-12 所示，单击“下一步(N)”按钮，在弹出的对话框中单击“是(Y)”按钮。



图 1-12　不使用 Oracle 自动安全更新

1.1.7　安装选项

此处有三个选项供使用者选择，如图 1-13 所示。在全新的环境下安装，前两个选项都是可以的，第三个选项用来对数据库软件进行升级。



图 1-13　选择创建和配置数据库

三个选项的区别如下（Oracle 数据库实例的概念请见相关章节）：

- 创建和配置数据库：安装数据库软件并创建一个数据库实例。
- 仅安装数据库软件：安装数据库软件，不创建数据库实例。
- 升级现有的数据库：升级低版本的 Oracle 数据库。

此处选择第一个选项，然后单击“下一步(N)”按钮进入下一个环节。

1.1.8　服务器类别

这一步确认安装类型。如果是安装到服务器上，请选中“服务器类”单选按钮。Oracle 数据库软件的安装已经非常人性化，通过服务器类别的区分，会自动匹配不同的配置要求。在个人测试学习环境中，可以选择“桌面类”，如图 1-14 所示。在桌面类别下，Oracle 数据库软件允许采用最低的系统配置。当然，如果你的电脑配置非常高，选择服务器类也可以，此处并不是要严格地按照功能定位来做区分，本质上，还是要看安装 Oracle 数据库软件的计算机配置是否足够。此处请选中“桌面类”单选按钮，单击“下一步(N)”按钮。



图 1-14　选择“桌面类”

1.1.9　Oracle 安装用户

这一步创建数据库软件安装用户，并按要求设置密码。从数据库安装的角度来考虑，建议使用非管理员账户来安装配置 Oracle 软件，与在 UNIX/Linux 上安装 Oracle 软件是一样的。在 Windows 平台不建议使用 administrator 用户，在 UNIX/Linux 平台上不建议使用 root 用户。当然，如果非要使用 root 用户也可以安装成功，但是这并不符合规范。

如图 1-15 所示，此处创建的用户就是安装、配置和管理数据库的用户，之后的所有软件操作都可以使用该用户。填写完毕之后，单击“下一步(N)”按钮。

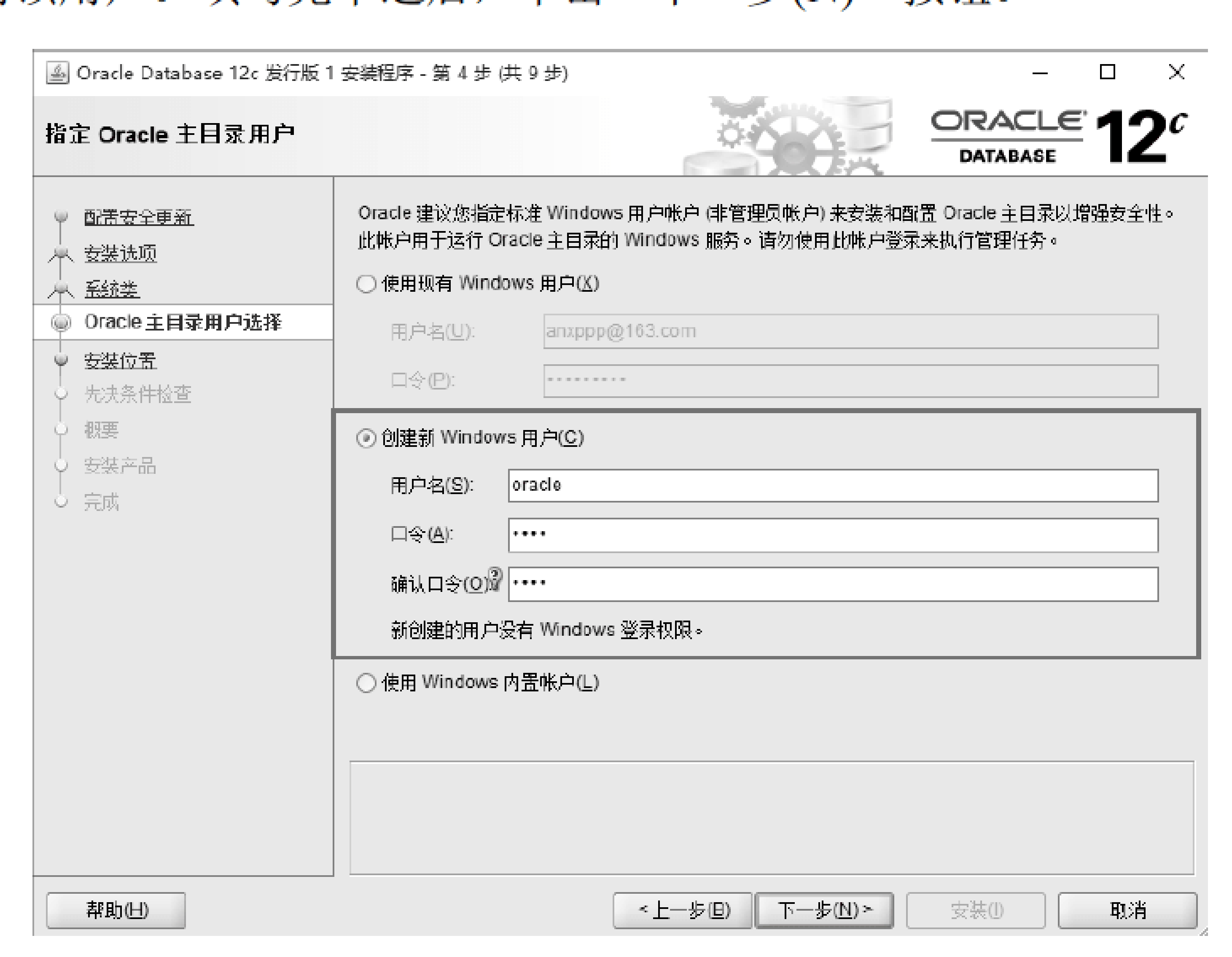


图 1-15　创建数据库软件安装用户和密码

三个选项的主要区别在于：

- 使用现有 Windows 用户：如果选中该项，就需要指定没有管理权限的用户。
- 创建新 Windows 用户：创建一个新用户，输入用户名和密码，确认密码。
- 使用 Windows 内置账户：内置账户，建议使用权限受限的用户。



无论是 Oracle 数据库软件，还是其他商业系统软件，标准合理的安装规范都是非常重要的。

1.1.10　安装位置

这一步填写数据库软件安装的目录、数据库名称等重要信息。如图 1-16 所示，此处需要填写的内容相对较多，主要是 Oracle 数据库软件的安装位置，包括“Oracle 基目录”和“软件位置”等。“Oracle 基目录”中存放所有 Oracle 相关软件的基本目录。“软件位置”也叫作 ORACLE_HOME 目录，用来存放 Oracle 数据库软件的目录。注意，ORACLE_HOME 目录和“基目录”是一种子集或者被包含的关系。“字符集”选 UTF-8（本书的 Oracle 数据库字符集统一配置为 UTF-8），然后是管理密码，如果密码设计太简单就会出现警告信息，不过可以忽略警告继续。其他有默认值的，也可以改为自己希望的那样。填写完毕之后，单击“下一步(N)”按钮。



图 1-16　填写数据库软件安装目录和数据库名称



此处有容器数据库的选项，关于容器数据库的相关概念，请见相关章节。此处为了安装展示，勾选了“创建为容器数据库”复选框。

1.1.11　先决条件、概要、安装产品

这一步是安装的最后阶段，完成软件的安装。先决条件是软件安装前的最后一次检查，安装程序会更加细致地检查操作系统平台相关的配置是否满足 Oracle 数据库软件安装的需求。在 Windows 系统上主要是计算机硬件配置方面，其内核参数方面无须做调整。在 UNIX/Linux 操作系统上就不同了，除了基础的硬件配置之外，安装程序会详细地检查系统内核参数配置、软件包的安装情况等。



对于初学者，难免在初次安装 Oracle 数据库软件时出现前期的准备工作不到位的情况，没有关系，在开始正式安装之前，Oracle 会对所有条件进行再次检查确认，当有不合理的地方时，程序会给出详细的提示和处理建议，只需要根据 Oracle 安装程序的提示再次修改即可。

概要的展示就是向安装者展示即将要安装的软件概况，之后就是正式的安装过程，如图 1-17 所示，安装程序开始部署程序文件，逐步完成数据库软件的安装。



图 1-17　完成数据库软件安装

根据计算机配置的不同，安装的速度也有明显的差异，安装到这个阶段时只需要等待安装进程执行完毕即可。

1.2 小结

通过本章的学习，读者可以了解 Oracle 数据库安装的整个过程，并且可以根据书中的提示自行完成软件安装的练习。数据库安装其实不是学习的难点，初期可能在安装中会遇到一些问题，不过大多数情况下都是由于操作系统环境或者某些地方配置不当所导致的，建议读者多安装几次软件，总结自己的安装脚本。

本章的内容可以当作对 Oracle 数据库的初体验，在下一章节中，我们将介绍 Oracle 数据库的基本概念，帮助读者进一步了解 Oracle 数据库。

第 2 章 Oracle数据库基本概念

本章主要向读者介绍 Oracle 数据库的基本概念，目的是让读者对什么是数据库或者 Oracle 数据库是什么样的数据库有一个初步的印象。在之后的章节中，本书将逐步深入地介绍 Oracle 数据库的各个细节部分。虽然 Oracle 数据库发展了很多年，技术也在不断更新，但是作为当下使用最广泛的数据库软件，Oracle 数据库的本质并没有发生变化。

2.1 关于关系数据库

每个组织都有其必须存储和管理的信息，以满足其需求。例如，公司必须收集和维护其雇员的人力资源记录。此信息对需要它的人必须是可用的。信息系统是一个正式的系统，用于存储和处理信息。信息系统可能是一组文件柜，其中包含许多文件夹，以及如何存储和检索文件夹的规则。但是，大多数公司现在使用数据库来自动化其信息系统。数据库是信息的一个有组织的集合，被作为一个整体来看待。数据库的目的是收集、存储和检索相关的信息，以供数据库应用程序使用。整个系统一般被称为 Database Management System（DBMS，数据库管理系统）。

通常，一个 DBMS 具有以下元素：

- 内核代码，此代码为 DBMS 管理内存和存储。
- 元数据的存储库，此存储库通常称为数据字典。
- 查询语言，此语言使应用程序能够访问数据。

实际上企业所使用的数据库泛指数据库应用程序，是一个与数据库进行交互，以访问和操作其数据的软件程序。第一代的数据库管理系统包括以下类型：

- 层次型：层次数据库把数据组织在树状结构中。每个父记录都有一个或多个子记录，类似于文件系统的结构。
- 网络型：网络数据库类似于层次数据库，但有一个区别，即记录之间是多对多的关系，而不是一对多的关系。

- 关系模型：E.F.Codd 在他 1970 年发表的论文《大型共享数据库数据的关系模型》中，定义了一个基于数学集合理论的关系模型。目前，最广泛接受的数据库模型就是关系模型。

关系数据库是一个符合关系模型的数据库。关系模型有以下主要方面：

- 结构：定义良好的对象，用于存储或访问数据库的数据。
- 操作：清楚定义的操作，使应用程序可以处理数据库中的数据和结构。
- 完整性规则：完整性规则用于管理在数据库中的数据和结构上的操作。

关系数据库实际上就是将数据存储在一组简单的关系中。关系是一个元组的集合，一个元组是一些属性值的无序集合。表是一个关系的二维表示，关系由行（元组）和列（属性）的形式构成。表中的每一行具有相同的列集。关系数据库是一个将数据存储在关系（表）中的数据库。例如，关系数据库可以在一个雇员表、部门表和薪金表中存储有关公司雇员的信息。

2.2 Oracle 数据库的发展史

Oracle 数据库的当前版本是超过 30 年的创新发展的结果。Oracle 数据库发展过程中的重要事件包括：

- 创立 Oracle 公司。1977 年，拉里 · 埃利森、鲍勃 · 穆勒、和爱德 · 奥茨成立了“软件开发实验室”咨询公司，其后又叫作“关系软件”（RSI）公司，1983 年，RSI 公司成为 Oracle 系统公司，再后来又成为 Oracle 公司。
- 第一个商用 RDBMS。1979 年，RSI 公司发布了 Oracle V2（版本 2），这是第一个商用的基于 SQL 的 RDBMS，它是关系数据库发展史中的一个里程碑事件。
- 可移植版本的 Oracle 数据库。Oracle 版本 3 发布于 1983 年，是第一个可以同时在大型机、小型机和个人电脑上运行的关系数据库。该数据库用 C 语言编写，使其可以被移植到多种平台上。
- 并发控制、数据分发和可扩展性等增强功能。版本 4 引入了多版本读一致性。版本 5 发布于 1985 年，支持客户端/服务器计算和分布式数据库系统。版本 6 增强了磁盘 I/O、行锁定、可扩展性以及备份和恢复。并且，版本 6 推出了 PL/SQL 语言第一版，这是专门针对 SQL 的过程化扩展。
- PL/SQL 存储程序单元。Oracle 7 发布于 1992 年，引入了 PL/SQL 存储过程和触发器。
- 对象和分区。

Oracle 8 发布于 1997 年，作为一种对象-关系数据库，支持许多新的数据类型。此外，Oracle 8 支持对大型表进行分区。

因特网计算，Oracle 8i 数据库发布于 1999 年，提供了互联网协议的原生支持和服务器端

的 Java 支持。Oracle 8i 是为因特网计算而设计的，这使得数据库可以在多层环境中部署。

Oracle 真正应用集群（Oracle RAC），Oracle 9i 数据库在 2001 年引入了 Oracle RAC，使得多个实例可以同时访问单个数据库。此外，Oracle XML 数据库（Oracle XML DB）引入了存储和查询 XML 的能力。

- 网格计算。Oracle 数据库 10g 在 2003 年引入了网格计算。此版本使得各个公司可以通过构建基于低成本服务器的网格基础设施来虚拟化计算资源。一个关键的目标是使数据库可以自我管理和自我优化。Oracle 自动存储管理（Oracle ASM）通过虚拟化和简化数据库存储管理，有助于实现这一目标。
- 可管理性、可诊断性和可用性。Oracle 数据库 11g，发布于 2007 年，引入了大量的新功能，使管理员和开发人员可以快速适应不断变化的业务需求。这种适应性的关键在于通过整合信息和尽可能使用自动化来简化信息基础架构。
- 2013 年 6 月 26 日 Oracle Database 12c 版本正式发布。与之前 10g、11g 里的 g 代表 grid 类似，12c 版本中的“c”是 cloud，代表云计算的意思。

2.3 认识数据库对象

常见的数据库对象包括表、索引、视图。

1. 表

对于初学者来说，对表的概念也有一定的认识。因为使用者对数据库的操作，90%以上是对表的操作。

什么是数据库表呢？它用于描述一个实体，例如雇员。可以使用一个表名（如 employees ）和一个列集来定义表。当用户创建表时，应该给出每一列的列名、数据类型和宽。

对于关系型数据来说，表其实就是许多行数据的集合。每条数据对应表中的一行。表中列用来标识实体的属性，而行用来标识实体的实例。例如，雇员实体的属性对应雇员 ID 列和姓氏列。行标识一个特定的雇员，可以选择性地为每个表列指定规则。这些规则称为完整性约束。例如，“非空”即是一个完整性约束。此约束强制每一行中的该列都包含一个值。

2. 索引

在关系数据库中，索引是一种与表有关的数据库结构，同时也是一个可选的数据结构，可以在表中的一个或多个列上创建索引。索引可以提高数据检索的性能。在处理一个请求时，数据库可以使用可用索引有效地找到请求的行。当应用程序经常查询某一特定行或特定范围的行时，索引很有用。

索引在逻辑和物理上都独立于数据。因此，可以删除和创建索引，而对表或其他索引没有任何影响。在删除索引后，所有应用程序可以继续运行。用户可以将索引的作用想象为一本图书的目录，根据目录中的页码快速找到所需的内容，所以在实际的企业生产环境中，索引的使

用是必不可少的。

当检索表中少量的行时，使用 Oracle 索引能够更快速地访问表中的这些行。索引存储了进行索引的列的值，同时存储包含索引值的行的物理 Rowid，唯一的例外是索引组织表（IOT），它使用主键作为逻辑 ROWID。一旦在索引中找到匹配值，索引中的 ROWID 就会指向表行的确切位置：哪个文件、文件中的哪个块以及块中的哪一行。可以在一列或多个列上创建索引。索引条目存储在 B-树结构中，因此遍历索引以找到行的键值只需要使用非常少的 O 操作。

在唯一索引的情况下，使用索引可能有两个目的：提高搜索行的速度，以及在索引列上实施唯一或主键约束。在插入、更新或删除表行的内容时，自动更新索引中的条目。删除表时，在该表上创建的所有索引也自动被删除。

3. 视图

视图允许用户查看单独表或多个连接表中数据的自定义表示。视图也称为“存储查询”，用户无法看到视图底层的查询细节。普通的视图不存储任何数据，而只存储定义，并且在每次访问视图时都运行底层的查询。

普通视图的扩展称为“物化视图”，允许同时存储查询的结果和查询的定义，从而加快处理速度，另外还有其他优点。对象视图类似于传统的视图，它可以隐藏底层表连接的细节，并且允许在数据库中进行面向对象的开发和处理，而底层的表仍然保持数据库关系表的格式。

2.4 表

比较常见的表类型有规则表（regular table），严格意义上来说又叫 heap table（堆表），这是最普通的一张表，其他类型还有 partition table、index-organized table、cluster 三种表类型，本节的重点就是讲解一些普通的表。

2.4.1 堆表

对于一张普通的表，它存放数据的规则是无序，假设把数据的存储空间看成学生宿舍楼一个连一个的房间，并不是第一个来的人就一定先在第一个房间。先来的人只要发现某个房间还有床位是空的就可以入住。

那么如何让他变成有序的呢？我可以专业创建一列来记录顺序。宿管在一楼门口发号码，进来一个同学，发一个号码，上面标注几号房间几号床位。这样所有入住的同学都是有序的。

堆表是数据库中最常见的表类型，以堆的形式进行组织。换句话说，表中的行没有按照任何特定的顺序存储，在 create table 命令中，可以指定子句来定义以堆表的形式组织的表，但是堆表属于默认值，所以一般用户在创建数据表时不需要添加这个关键字。在堆表中，每一行包含一列或者多列，每一列都有一种数据类型和一个长度，从 Oracle 8i 版本开始，列也可以包含用户定义的对象类型。

【示例 2-1】下面举一个简单的例子。

```
SQL> create table t
 (a int,
 b varchar2(50),
 c varchar2(50));          //创建一个表

Table created.

insert into t(a) values(1);
insert into t(a) values(2);
insert into t(a) values(3);
insert into t(a) values(4);
insert into t(a) values(5);
insert into t(a) values(6);

insert into t(b) values(111);
insert into t(b) values(222);
insert into t(b) values(222);
insert into t(b) values(333);
insert into t(b) values(444);.........
select a from t;

       A
----------
       1
       2
       3
       4
       5
       6//上面查询插入的结果是无序的，如何变成的有序的呢？加上 order by
SQL>
select a from t order by a desc;

       A
----------

       6
       5
       4
       3
       2
       1
```

创建一个较为复杂的单表：

```
CREATE TABLE employee
      (EMPNO NUMBER(4) NOT NULL,
       ENAME VARCHAR2(10),
       JOB VARCHAR2(9),
       MGR NUMBER(4),
```

```
       HIREDATE DATE,
       SAL NUMBER(7, 2),
       COMM NUMBER(7, 2),
       DEPTNO NUMBER(2));

INSERT INTO employee VALUES
       (7369, 'SMITH',  'CLERK',     7902,
       TO_DATE('17-11-1980', 'DD-MM-YYYY'),  800, NULL, 20);
INSERT INTO employee VALUES
       (7499, 'ALLEN',  'SALESMAN',  7698,
       TO_DATE('20-11-1981', 'DD-MM-YYYY'), 1600,  300, 30);
INSERT INTO employee VALUES
       (7521, 'WARD',   'SALESMAN',  7698,
       TO_DATE('22-11-1981', 'DD-MM-YYYY'), 1250,  500, 30);
INSERT INTO employee VALUES
       (7566, 'JONES',  'MANAGER',   7839,
       TO_DATE('2-11-1981', 'DD-MM-YYYY'),  2975, NULL, 20);
INSERT INTO employee VALUES
       (7654, 'MARTIN', 'SALESMAN',  7698,
       TO_DATE('28-11-1981', 'DD-MM-YYYY'), 1250, 1400, 30);
INSERT INTO employee VALUES
       (7698, 'BLAKE',  'MANAGER',   7839,
       TO_DATE('1-11-1981', 'DD-MM-YYYY'),  2850, NULL, 30);
INSERT INTO employee VALUES
       (7782, 'CLARK',  'MANAGER',   7839,
       TO_DATE('9-11-1981', 'DD-MM-YYYY'),  2450, NULL, 10);
INSERT INTO employee VALUES
       (7788, 'SCOTT',  'ANALYST',   7566,
       TO_DATE('09-11-1982', 'DD-MM-YYYY'), 3000, NULL, 20);
INSERT INTO employee VALUES
       (7839, 'KING',   'PRESIDENT', NULL,
       TO_DATE('17-11-1981', 'DD-MM-YYYY'), 5000, NULL, 10);
INSERT INTO employee VALUES
       (7844, 'TURNER', 'SALESMAN',  7698,
       TO_DATE('8-11-1981', 'DD-MM-YYYY'),  1500,    0, 30);
INSERT INTO employee VALUES
       (7876, 'ADAMS',  'CLERK',     7788,
       TO_DATE('12-11-1983', 'DD-MM-YYYY'), 1100, NULL, 20);
INSERT INTO employee VALUES
       (7900, 'JAMES',  'CLERK',     7698,
       TO_DATE('3-10-1981', 'DD-MM-YYYY'),   950, NULL, 30);
INSERT INTO employee VALUES
       (7902, 'FORD',   'ANALYST',   7566,
       TO_DATE('3-11-1981', 'DD-MM-YYYY'),  3000, NULL, 20);
INSERT INTO employee VALUES
       (7934, 'MILLER', 'CLERK',     7782,
       TO_DATE('23-12-1982', 'DD-MM-YYYY'), 1300, NULL, 10);
```

2.4.2 临时表

临时表就是用来暂时保存临时数据（或叫中间数据）的一个数据库对象，和普通表有些类似，然而又有很大区别。它只能存储在临时表空间，而非用户的表空间。Oracle 临时表是会话或事务级别的，只对当前会话或事务可见。每个会话只能查看和修改自己的数据。

目前所有使用 Oracle 作为数据库支撑平台的应用大部分都是数据量比较庞大的系统，即一般情况下都是在百万级以上的数据量。当然在 Oracle 中创建分区是一种不错的选择，但是当发现应用有多张表关联并且这些表大部分都比较庞大，在关联的时候会发现其中的某一张或者某几张表关联之后得到的结果集非常小并且查询得到这个结果集的速度非常快，那么这个时候就应考虑在 Oracle 中创建“临时表”了。

对临时表的概念也可以这样理解，在 Oracle 中创建一张表，这个表不用于其他的什么功能，主要用于自己的软件系统一些特有功能，用完之后表中的数据就没用了。Oracle 的临时表创建之后基本不占用表空间，如果没有指定临时表（包括临时表的索引）存放的表空间时，插入到临时表的数据是存放在 Oracle 系统的临时表空间（Temp）中的。

Oracle 临时表有两种类型：会话级的临时表和事务级的临时表。

1. ON COMMIT DELETE ROWS

它是临时表的默认参数，表示临时表中的数据仅在事务（transaction）过程中有效，当事务提交（commit）后，临时表的暂时段将被自动截断（truncate），但是临时表的结构以及元数据还存储在用户的数据字典中。在临时表完成它的使命后，最好将其删除，否则数据库会残留很多临时表的表结构和元数据。

会话级的临时表的数据和当前会话有关系，当前 SESSION 不退出的情况下，临时表中的数据就还存在，临时表的数据只有退出当前 SESSION 时才被截断（truncate table）。

【示例 2-2】会话级别的临时表创建

```
create global temporary table tmp test(id number,name varchar2(32)) on commit
preserve rows;
```

2. ON COMMIT PRESERVE ROWS

它表示临时表的内容可以跨事务而存在，不过，当该会话结束时，临时表的暂时段将随着会话的结束而被丢弃，临时表中的数据自然也就随之丢弃，但是临时表的结构以及元数据还存储在用户的数据字典中。在临时表完成它的使命后，最好将其删除，否则数据库会残留很多临时表的表结构和元数据。

事务级的临时表（默认）与事务有关，当进行事务提交或者事务回滚时，临时表的数据将自行截断，即当 commit 或 rollback 时，数据就会被截断，其他的特性和会话级的临时表一致。

【示例 2-3】事务级临时表的创建方法

```
create global temporary table tmp_test(id number,name varchar2(32)) on commit
delete rows;
```

2.4.3 索引组织表

索引组织表（index organized table，IOT）就是存储在一个索引结构中的表。

创建索引，可以更有效地查找表中的特定行，然而创建索引将带来额外的一些系统开销，因为数据库必须同时维护表的数据行和索引条目。如果表包含的列并不是很多，并且对表的访问主要集中在某一行上，那么应该怎么做呢？

在这种情况下索引组织表可能就是正确的解决方案。索引组织表是以 b 树索引的形式存储表中的行，其中 b 树索引的每个节点都包含作为键的列以及一个或多个非索引列。索引组织表最明显的优点在于只需要维护一个存储结构，而不是两个。类似的，表中主键的值只在索引组织表中存储一次，而在普通表中则需要存储两次。使用索引组织表也有一些缺点，有些表，例如记录事件的表可能不需要主键，或者在某些情况下不需要任何键，而索引组织表则必须有主键，同时索引组织表不可以是集群的成员。最后，如果表中有大量的列并且在检索表中的行时需要频繁地访问许多列，那么索引组织表可能就不是最佳的解决方案。

那么到底 IOT 表有什么意义呢？

使用堆组织表时，用户必须为表和表主键上的索引分别留出空间。IOT 不存在主键的空间开销，因为索引就是数据，数据就是索引，二者已经合二为一。但是，IOT 带来的好处并不止于节约了磁盘空间的占用，更重要的是大幅度降低了 I/O，减少了访问缓冲区缓存。尽管从缓冲区缓存获取数据比从硬盘读要快得多，但缓冲区缓存并不免费，而且也绝对不是廉价的。每个缓冲区缓存获取都需要缓冲区缓存的多个闩，而闩是串行化设备，会限制应用的扩展能力。

IOT 适用的场合有：

- 完全由主键组成的表。这样的表如果采用堆组织表，则表本身完全是多余的开销，因为所有的数据全部同样也保存在索引里，此时，堆表是没用的。
- 代码查找表，如果只会通过一个主键来访问一个表，这个表就非常适合实现为 IOT。
- 如果想保证数据存储在某个位置上，或者希望数据以某种特定的顺序物理存储，IOT 就是一种合适的结构。

IOT 提供如下好处：

- 提高缓冲区缓存效率，因为给定查询在缓存中需要的块更少。
- 减少缓冲区缓存访问，这会改善可扩缩性。
- 获取数据的工作总量更少，因为获取数据更快。
- 每个查询完成的物理 I/O 更少，因为对于任何给定的查询，需要的块更少，而且对地址记录的一个物理 I/O 很可能可以获取所有地址（而不只是其中一个地址，但堆表实现就只是获取一个地址）。如果经常在一个主键或唯一键上使用 BETWEEN 查询也是如此，因为相近的记录存在一起，查询时引入的逻辑 IO 和物理 IO 都会更少。

2.4.4 集群表

如果经常同时访问两个或多个表，例如一个订单表和一个项目表，那么创建集群表可能是

一个较好的方法，它可以改进应用这些表的查询性能。

在具有相关行是项目表和订单表共同拥有时，订单表和项目表的相关信息可以存储在同一个数据块中，从而减少检索的 IO 开销，还可以减少存储两个表共有的列所需的存储空间。

两个表共有的列称为集群键值。集群键值存储在集群索引中，针对集群索引的操作，非常类似于传统的索引。通过集群键值访问集群表时，可以改进对集群表的查询效率，共同的列只需存储一次，而不必针对每个行重复存储。相对于表执行的查询语句数量，如果需要频繁地对表执行插入更新和删除操作，则集群表的优点会减弱，此外经常对集群中的单个表进行查询，集群表的特点也不会得到更好的体现。

2.4.5　分区表

对表进行分区，或对索引进行分区，可帮助建立更加易于管理的大型表。可以将表分区为较小的部分，从应用程序的观点来看，分区是透明的。也就是说，在终端用户的 SQL 中不需要对任何特定分区进行显式的引用，用户唯一能够观察到的是在 WHERE 子句后面使用符合分区方案的筛选条件。对分区表进行查询，用户会发现 SQL 运行得更为快速，从 DBA 的角度看，对表进行分区有很多优点，如果表的一个分区位于已损坏的磁盘卷上，用户仍然可以查询表的其他分区。



DBA 是 Database Administrator 的英文缩写，即数据库管理员。

分区有三种类型：范围分区、散列分区以及从 Oracle 9i 开始引入的列表分区。从 Oracle 11g 开始，也可以根据父与子的关系进行分区，可以由应用程序控制分区，并且可以对基本分区类型进行很多组合，包括列表-散列、列表-列表、列表-范围和范围-范围等组合分区类型，分区表中的每一行只能存在于一个分区中，分区键用于对行数据指定正确的分区，分区键可以是组合键，最多可组合表中的 16 个列。

一般来说，推荐对于任何大于 2GB 的表，应尽量考虑对其进行分区并且表中包含历史数据，新的数据被增加到新的分区中。

表分区的优点：

- 改善查询性能：对分区对象的查询可以仅搜索自己关心的分区，提高检索速度。
- 增强可用性：如果表的某个分区出现故障，表在其他分区的数据仍然可用。
- 维护方便：如果表的某个分区出现故障，需要修复数据，只修复该分区即可。
- 均衡 I/O：可以把不同的分区映射到磁盘以平衡 I/O，改善整个系统性能。

下面罗列表分区的几种类型及操作方法。

1. 范围分区

范围分区将数据基于范围映射到每一个分区，这个范围是你在创建分区时指定的分区键决定的。这种分区方式是最为常用的，并且分区键经常采用日期。例如，你可能会将销售数据按

照月份进行分区。

当使用范围分区时，请考虑以下几个规则：

- 每一个分区都必须有一个 VALUES LESS THEN 子句，它指定了一个不包括在该分区中的上限值。分区键的任何值等于或者大于这个上限值的记录都会被加入下一个高一些的分区中。
- 除了第一个分区，其他分区都会有一个隐式的下限值，这个值就是此分区的前一个分区的上限值。
- 在最高的分区中，MAXVALUE 被定义。MAXVALUE 代表了一个不确定的值。这个值高于其他分区中任何分区键的值，也可以理解为高于任何分区中指定的 VALUE LESS THEN 的值，同时包括空值。

假设有一个 CUSTOMER 表，表中有数据 200000 行，将此表通过 CUSTOMER_ID 进行分区，每个分区存储 100000 行，将每个分区保存到单独的表空间中，这样数据文件就可以跨越多个物理磁盘了。

【示例 2-4】创建表和分区

```
create table customer
(
   customer id number not null primary key,
   first name  varchar2(30) not null,
   last_name   varchar2(30) not null,
   phonevarchar2(15) not null,
   emailvarchar2(80),
   status      char(1)
)
partition by range (customer id)
(
   partition cus_part1 values less than (100000) tablespace cus_ts01,
   partition cus_part2 values less than (200000) tablespace cus_ts02
);
```

【示例 2-5】按时间分区

```
create table sales (
 product_id varchar2(5)
 , sales date date
 , sales cost number(10)
 , status varchar2(20)
 ) partition by range (sales_date) subpartition by list (status) (
 partition p1 values less than(to_date('2003-01-01', 'yyyy-mm-dd')) tablespace
rptfact2009(subpartition p1sub1 values ('active') tablespace rptfact2009,
subpartition p1sub2 values ('inactive') tablespace rptfact2009)
 , partition p2 values less than(to_date('2003-03-01', 'yyyy-mm-dd')) tablespace
rptfact2009(subpartition p2sub1 values ('active') tablespace rptfact2009,
subpartition p2sub2 values ('inactive') tablespace rptfact2009)
 );
```

2. 列表分区

列表分区的特点是某列的值只有几个，基于这样的特点可以采用列表分区。

【示例 2-6】列表分区

```
create table problem tickets
(
   problem_id   number(7) not null primary key,
   description  varchar2(2000),
   customer_id  number(7) not null,
   date entered date not null,
   status       varchar2(20)
)
partition by list (status)
(
     partition prob_active   values ('active') tablespace prob_ts01,
     partition prob inactive values ('inactive') tablespace prob ts02
);
```

3. 散列分区

这类分区是在列值上使用散列算法，以确定将行放入哪个分区中。当列的值没有合适的条件时，建议使用散列分区。散列分区为通过指定分区编号来均匀分布数据的一种分区类型，因为通过在 I/O 设备上进行散列分区，使得这些分区大小一致。

【示例 2-7】散列分区

```
create table hash table
(
  col number(8),
  inf varchar2(100)
)
partition by hash (col)
(
  partition part01 tablespace hash_ts01,
  partition part02 tablespace hash_ts02,
  partition part03 tablespace hash_ts03
);
```

散列分区最主要的机制是根据 hash 算法来计算具体某条记录应该插入哪个分区中，hash 算法中最重要的是 hash 函数，Oracle 中如果要使用 hash 分区，只需指定分区的数量即可。建议分区的数量采用 2 的 n 次方，这样可以使各个分区间数据分布更加均匀。

4. 组合范围散列分区

这种分区是基于范围分区和列表分区的，表首先按某列进行范围分区，然后按某列进行列表分区，分区之中的分区被称为子分区。

【示例 2-8】组合范围散列分区

```
create table sales (
 product id varchar2(5)
 , sales_date date
 , sales_cost number(10)
 , status varchar2(20)
 ) partition by range (sales_date) subpartition by list (status) (
 partition p1 values less than(to date('2003-01-01', 'yyyy-mm-dd')) tablespace
rptfact2009(subpartition p1sub1 values ('active') tablespace rptfact2009,
subpartition p1sub2 values ('inactive') tablespace rptfact2009)
 , partition p2 values less than(to_date('2003-03-01', 'yyyy-mm-dd')) tablespace
rptfact2009(subpartition p2sub1 values ('active') tablespace rptfact2009,
subpartition p2sub2 values ('inactive') tablespace rptfact2009)
 );
```

5. 复合范围散列分区

这种分区是基于范围分区和散列分区的，表首先按某列进行范围分区，然后按某列进行散列分区。

【示例 2-9】复合范围散列分区

```
CREATE TABLE RANGE_HASH_TEST (
 TRANSACTION_ID NUMBER PRIMARY KEY
 ,ITEM_ID NUMBER(8) NOT NULL
 ,ITEM_DESCRIPTION VARCHAR2(300)
 ,TRANSACTION DATE DATE
 ) PARTITION BY RANGE (TRANSACTION DATE) SUBPARTITION BY HASH (TRANSACTION ID)
SUBPARTITIONS 3 STORE IN (
 DINYA_SPACE01
 ,DINYA_SPACE02
 ,DINYA_SPACE03
 ) (
 PARTITION PART 01 VALUES LESS THAN(TO DATE('2006-01-01', 'YYYY-MM-DD'))
 ,PARTITION PART_02 VALUES LESS THAN(TO_DATE('2010-01-01', 'YYYY-MM-DD'))
 ,PARTITION PART_03 VALUES LESS THAN(MAXVALUE)
 );
```

2.5 索引

Oracle 中有一些可用的索引类型，每种索引都适合于特定的表类型、访问方法或应用程序环境。下面几个小节将介绍最常见的索引类型的重点内容和特性。

（1）唯一索引。唯一索引是最常见的 B 树索引形式，经常用于实施表的主键约束，其作用在于确保索引列中不存在重复的值。可以在 T 表中 a 列上创建唯一索引。

（2）非唯一索引。非唯一索引帮助提高表访问的速度，而不会实施唯一性。例如，可以

在 t 表的 b 列上创建非唯一索引，从而提高按姓查找的速度。但是，对于任何给定的值，确实可以有许多重复的值，如果在 create index 语句中没有指定其他任何关键字，则默认在列上创建非唯一 B 树索引。

（3）反向键索引。反向键索引是特殊类型的索引，一般用于 OLTP（联机事务处理）环境中。在反向键索引中，反向每个列的索引键值中的所有字节。在 create index 命令中，使用 reverse 关键字指定反向键索引。

【示例 2-10】创建反向键索引

```
create index index_reverse on t(a) reverse;
```

插入到表中的内容分布在索引的所有叶键上，从而减少一些插入新行的写入程序之间的争用，如果在发出订单后不久就查询或修改订单，则反向键索引也可减少 OLTP 环境中这些“热点”的潜在性。

（4）基于函数的索引。基于函数的索引类似于标准的 B 树索引，不同之处在于它将被声明为表达式的列的变换形式存储在索引中，而不是存储列自身。当名称和地址可以作为混合内容存储在数据库中时，基于函数的索引就非常有用了，如果搜索标准是“Smith”，在包含值“Smith”的列上进行普通索引就不会返回任何值。另一方面，如果索引以全大写字母的形式存储姓，那么所有对姓的搜索都可以使用大写字母。

【示例 2-11】在 employee 表的 ename 列上创建基于函数的索引

```
create index  up_name on employee(upper(ename));
```

因此，使用如下查询的搜索将使用前面所创建的索引，而不是进行完整的表扫描。

【示例 2-12】使用索引

```
select EMPNO,ename from employee where upper(ename)= 'SMITH';

    EMPNO ENAME
---------- ----------
     7369 SMITH
```

（5）位图索引。在索引的叶节点上，位图索引结构与 B 树索引存在着较大的区别。它只存储索引列每个可能值（基数）的一个位串，位串的长度与索引表中的行数相同。与传统索引相比，位图索引除了可以节省大量的空间外，还可以大大缩短响应时间，因为在需要访问表自身之前，Oracle 就可以从包含多个 where 子句的查询中快速删除潜在的行。对于多个位图可以使用逻辑 and 和 or 操作来确定访问表中的哪些行。虽然位图索引可用于表中的任何列，但在索引列具有较低基数或大量不同的值时，使用位图索引才最有效。例如，PERS 表中的 Gender 列将有 NULL、M 或 F 值。

Gender 列上的位图索引将只有 3 个位图存储在索引中。另一方面，last name 列上的位图索引将有和表中行数基本相同的位图串数量。如果执行完整的表扫描而不是使用索引，则查找特定姓的查询将很可能花费较少的时间。在这种情况下，使用传统的 B 树非唯一索引将更有

意义。位图索引的一种变体称为“位图连接索引”，这种索引在某个表列上创建一个位图索引，此列常常根据相同的列与一个或多个其他的表相连接。这就在数据仓库环境中提供了大量的优点，在一个事实表和一维或多维表上创建位图连接索引，实质上等于预先连接这些表，从而在执行实际的连接时节省 CPU 和 IO 资源。

现在简单总结几点索引的作用：

- 索引是数据库对象之一，用于加快数据的检索，类似于书籍的索引。在数据库中索引可以减少数据库程序查询结果时需要读取的数据量，类似于在书籍中利用索引可以不用翻阅整本书即可找到想要的信息。
- 索引是建立在表上的可选对象；索引的关键在于通过一组排序后的索引键来取代默认的全表扫描检索方式，从而提高检索效率。
- 索引在逻辑上和物理上都与相关的表和数据无关，当创建或者删除一个索引时，不会影响基本的表。
- 索引一旦建立，在表上进行 DML 操作时（例如在执行插入、修改或者删除相关操作时），Oracle 会自动管理索引，索引删除，不会对表产生影响。
- 索引对用户是透明的，无论表上是否有索引，SQL 语句的用法不变。
- Oracle 创建主键时会自动在该列上创建索引。

2.5.1 索引的使用

1. 创建索引

【示例 2-13】创建索引

```
create [unique] | [bitmap] index index_name  --unique 表示唯一索引
on table name([column1 [asc|desc], column2    --bitmap，创建位图索引
[asc|desc], …] | [express])
[tablespace tablespace_name]
[pctfree n1]                                  --指定索引在数据块中空闲空间
[storage (initial n2)]
[nologging]                                   --表示创建和重建索引时允许对表做 dml 操作，默认
情况下不应该使用
[noline]
[nosort];                                     --表示创建索引时不进行排序，默认不适用，如果数据
已经是按照该索引顺序排列的可以使用
```

2. 修改索引

【示例 2-14】重命名索引

```
alter index up_name rename to low_name;
```

【示例 2-15】合并索引（表使用一段时间后在索引中会产生碎片，此时索引效率会降低，可以选择重建索引或者合并索引，合并索引方式更好些，无须额外存储空间，代价较低）

```
alter index up_name coalesce;
```

【示例 2-16】重建索引

```
alter index up_name rebuild;
```

3. 删除索引

【示例 2-17】删除索引

```
drop index up_name;
```

4. 查看索引

【示例 2-18】查看索引

```
select index name,index-type, tablespace name, uniqueness from all indexes where
table_name =' EMPLOYEE ';
```

2.5.2　索引建立的原则

事物都有两面性，一方面索引可以提高查询效率，另一方面，Oracle 数据库自动维护索引同时消耗数据库性能。

用户在创建索引时也有一些事项需要注意：

（1）如果有两个或者以上的索引，其中有一个唯一性索引，而其他是非唯一，这种情况下 Oracle 将使用唯一性索引而完全忽略非唯一性索引，至少要包含组合索引的第一列（即如果索引建立在多个列上，只有它的第一个列被 where 子句引用时，优化器才会使用该索引）；

（2）限制表中索引的数量。

- 创建索引耗费时间，并且随数据量的增大而增大。
- 索引会占用物理空间。
- 当对表中的数据进行增加、删除和修改时，索引也要动态维护，降低了数据的维护速度。

2.6　视图

下面各小节将介绍一般数据库用户、开发人员或 DBA 创建并使用基本视图的基础知识。

2.6.1　普通视图

通常称之为“视图”，不会占据任何内存空间，只有它的定义（查询）存储在数据字典中。视图底层查询的表称为“基表”，视图中的每个基表都可以进一步定义为视图。

视图有许多优点，它可以隐藏数据复杂性：高级分析人员可以定义包含 EMPLOYEE 表的视图，这样上层管理部门可以更容易地使用 select 语句检索相关信息，这种检索表面上看起来是使用表，但实际上是包含查询的视图，该查询连接 EMPLOYEE 表。视图也可以用于实施安

全性。 EMPLOYEE 表上的视图 EMPINFO 只能检索雇员名和雇员编号，并且该视图应定义为只读，从而防止更新该表。

【示例 2-19】创建普通视图

```
create view empinfo as select EMPNO, ENAME from employee with read only;
```

如果没有 read only 子句，则可以更新某行或添加行到视图中，甚至在包含多个表的视图上进行这些操作。视图中有一些构造可防止对其进行更新，例如使用 distinct 操作符、聚集函数或 group by 子句。当 Oracle 处理包含视图的查询时，它替换用户 select 语句中的底层查询定义，并且处理结果查询，就好像视图不存在一样。因此，在使用视图时，基表上任何已有索引的优点并没有改变。

2.6.2 物化视图

在某些方面，物化视图非常类似于普通视图：视图的定义存储在数据字典中，并且该视图对用户隐藏底层基查询的细节。但是，相似之处仅限于此。

物化视图也在数据库段中分配空间，用于保存执行基查询得到的结果集。物化视图可用于将表的只读副本复制到另一个数据库，该副本具有和基表相同的列定义和数据。这是物化视图最简单的实现。为了减少刷新物化视图时的响应时间，可以创建物化视图日志以刷新物化视图。否则，在需要刷新时就必须进行完全刷新，即必须获取基查询的全部结果以刷新物化视图。

物化视图日志为以增量方式更新物化视图提供了方便。在数据仓库环境中，物化视图可存储来自于 group by rollup 或 group by cube 查询的聚集数据，如果设置适当的初始参数值，例如 query rewrite enable，并且查询自身允许查询重写（使用 query rente 子句），则任何与物化视图执行相同类型的聚集操作的查询将自动使用物化视图，而不是运行初始的查询。无论物化视图的类型是什么，在基表中提交事务或根据需要刷新它时，系统都会自动对物化视图进行刷新。

物化视图在很多方面类似于索引，它们都直接和表联系并且占用空间，在更新基表时必须刷新它们，它们的存在对用户而言实际上是透明的。通过使用可选的访问路径来返回查询结果，它们可以帮助优化查询。

2.6.3 对象视图

面向对象（OO）的应用程序开发环境已经变得越来越流行，Oracle 10g 数据库完全支持数据库中本地化对象和方法的实现。然而，从纯粹的关系数据库环境向纯粹的 OO 数据库环境迁移并不是容易的过程，很少有组织愿意花费时间和资源从头开始构建新的系统，而 Oracle 10g 使用对象视图使这种变迁变得更为容易。

对象视图允许面向对象的应用程序查看作为对象集合的数据，这种对象集合具有属性和方法，而遗留系统仍然可以对 INVENTORY 表运行批处理作业。对象视图可以模仿抽象数据类型、对象标识符（OID）以及纯粹的 OO 数据库环境能够提供的引用。和普通视图一样，可以

在视图定义中使用 instead of 触发器来允许针对视图的 DML，这里使用的是 pl / sql 代码块，而不是用户或应用程序提供的实际 DML 语句。

2.7 小结

本章主要介绍了 Oracle 数据库的基本概念和一般的数据库对象，这部分的内容其实不单单是针对 Oracle 数据库，表或者索引的概念在各类关系型数据库中都是很重要的基本概念，作为数据库管理人员，日常需要处理很多与表和索引相关的工作，需要读者重点掌握。通过章的学习，对数据库有了初步的认识，在下一章中，将重点介绍 Oracle 数据库的体系结构，这部分的内容会是学习 Oracle 数据库的重中之重，在 DBA 的生涯中，所有的知识点都会是围绕其体系结构展开的。

第 3 章

Oracle数据库体系结构

对于一门技术的学习，尤其是像 Oracle Database 这种知识体系极其庞杂的技术来讲，从宏观上了解其体系结构是至关重要的。同时，未必是专业 DBA 人员才需要了解其体系结构（对于数据库专业人员来讲，这些固然是必备知识了），一般的技术人员如果对其有较深入的了解，也是大有益处的，毕竟技术思想很多时候都是相通的。

大多数的 Oracle 数据库使用者在实际工作中经常会遇到下面几个问题：

- 通常说 Oracle 数据库是什么？
- 如何理解 Oracle 实例？
- Oracle 实例由哪些部分组成，它们之间的作用是什么？
- 如何理解 Oracle 的物理结构？
- Oracle 的物理结构由哪些部分组成，它们之间的作用是什么？
- 如何理解 Oracle 的逻辑结构？
- Oracle 的逻辑结构由哪些部分组成，它们之间的作用是什么？

本章就从不同维度，如 Oracle 的内存结构、进程结构、存储结构等方面做相应描述。以上疑惑，读者可以从本章的学习中得到答案。

3.1 体系结构概述

众所周知，Oracle Database 是一款关系型数据库管理系统，同类的产品还有 MySQL、SQL Server 等。很多时候，多数人会把那个承载核心数据的系统笼统地称为数据库服务器，但从严格意义上来讲 Oracle 数据库是指 Oracle 数据库服务器（Oracle Server），由 Oracle 实例（Oracle Instance）和 Oracle 数据库（Oracle Database）组成。

（1）实例是数据库启动时初始化的一组进程和内存结构。实例启动时，系统首先在服务器内存中分配系统全局区（System Global Area），构成 Oracle 内存结构，然后启动必需的常驻内存的操作系统进程，组成 Oracle 的进程结构，内存结构和进程结构即构成 Oracle 实例。

（2）数据库指的是用户存储数据的一些物理文件，包括数据文件、重做日志文件、控制

文件、参数文件、密码文件、归档日志文件、备份文件、告警日志文件、跟踪文件等。其中，数据文件、控制文件、重做日志文件和参数文件是必需的，其他文件可选。一个实例只能对应/操作一个数据库，一个数据库可以由一个或多个实例操作（比如 RAC）。

从实例和数据库的概念上来看，实例是暂时的，不过是一组逻辑划分的内存结构和进程结构，会随着数据库的关闭而消失，而数据库其实就是一堆物理文件（控制文件、数据文件、日志文件等），它是永久存在的（除非磁盘损坏）。数据库和实例通常是一对一的，这种结构称为单实例体系结构。当然还有一些复杂的分布式结构，一个数据库可以对多个实例，像 Oracle 的 RAC。

从图 3-1 中可以清楚地看到每个组件之间的关联情况，这些组件包括内存组件和物理文件部分，组成了全部的 Oracle Database Server。

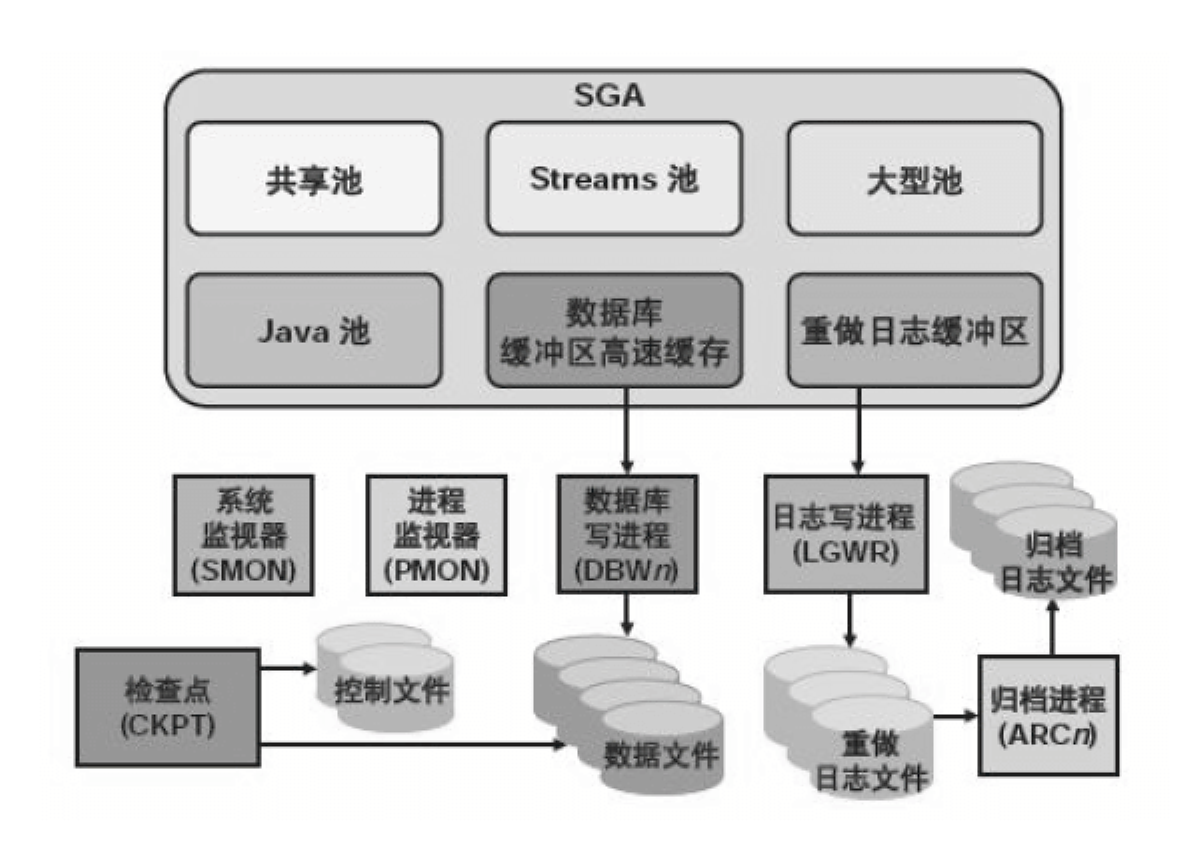


图 3-1　Oracle 数据库体系结构示意图

3.2　Oracle 数据库的连接

连接和会话都与用户进程密切相关，但意义却大不相同。

连接是用户进程和 Oracle DB 实例之间的通信路径。通信路径是使用可用的进程间通信机制（在一台同时运行用户进程和 Oracle DB 的计算机上）或网络软件（多台不同的计算机运行数据库应用程序和 Oracle DB 并通过网络进行通信时）建立的，如图 3-2 所示。

会话代表登录到数据库实例的当前用户的状态。例如，当某个用户启动 SQL*Plus 时，该用户必须提供有效的用户名和口令，然后系统会为该用户建立一个会话。会话从用户建立连接时开始，一直持续到用户断开连接或退出数据库应用程序时为止。

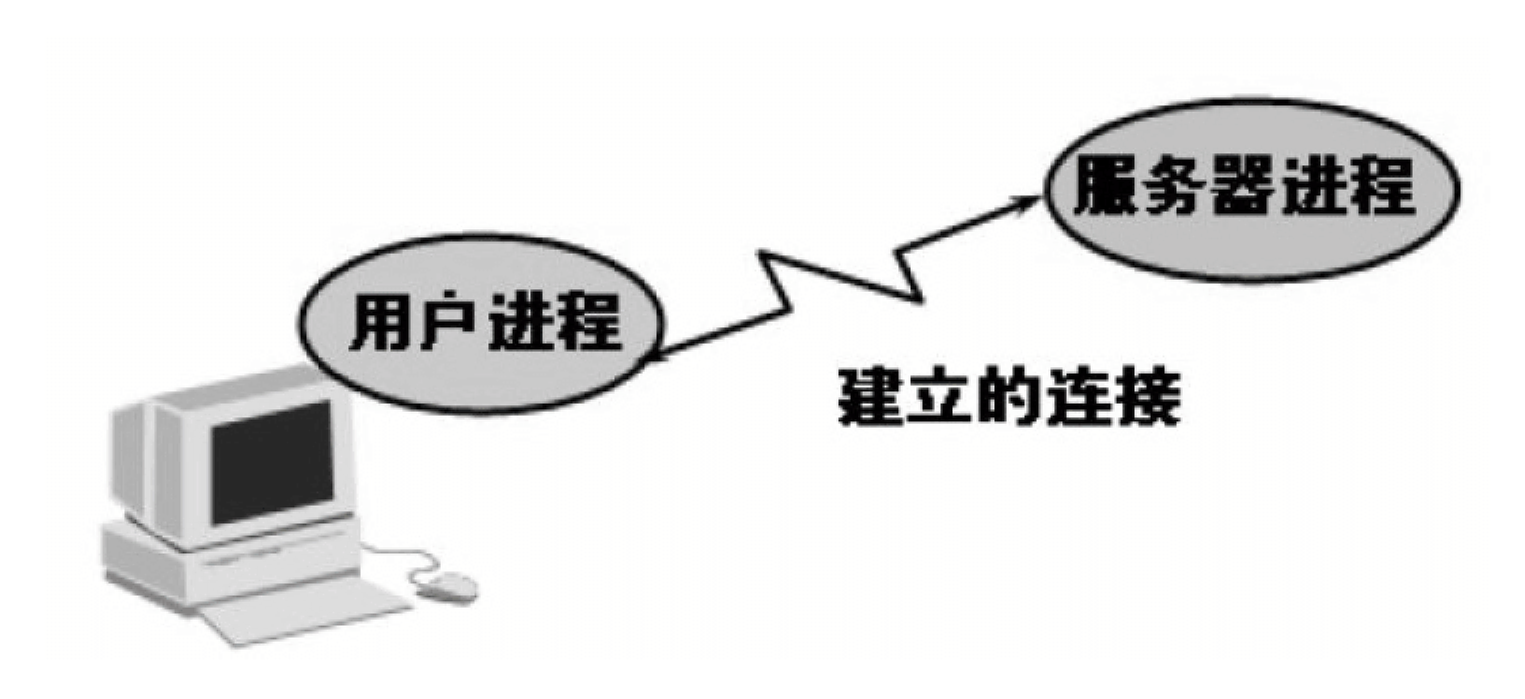


图 3-2　用户进程连接示意图

一个 Oracle DB 用户可以使用相同用户名创建多个会话，并让这些会话并存。例如，用户名/口令为 HR/HR 的用户可以多次连接到同一个 Oracle DB 实例。用户进程可以是一般的客户端软件，像 Oracle 的 sqlplus、sql developer，或者是一些驱动程序等都属于用户进程。

服务器进程有时会称为前台进程，当然是相对于后台进程（后面会提到数据库写入器、日志写入器等）来说的，服务器进程的主要作用就是处理连接到当前实例的用户进程的请求，对客户端发来的 sql 进行执行并返回执行结果。在专有服务器结构中，用户进程和服务器进程是一对一的，也就是说，当监听程序监听到客户端来了一个请求，会为其分配一个对应的服务器进程。还有一种结构为共享服务器，这种结构就不是一个用户进程对应一个服务器进程了，会通过调度程序进行协调处理，关于共享服务器连接，本文就不再赘述了。

上面描述了一些在进行数据库连接操作时大致的交互流程是什么样的。下面就来看看 Oracle 的实例内存结构。

3.3 实例内存区

由于内存结构和进程结构关系较紧密，进程会作用到对应的内存区域，比如数据库写入器作用到数据库缓冲区缓存中，日志写入器会作用到日志缓冲区，所以内存结构和进程结构会相互配合地进行描述。

Oracle 实例内存结构由 SGA（系统全局区）和 PGA（用户全局区）两部分组成。SGA 是一块共享的内存区域，也是最大的一块内存区域；PGA 则是用户会话专有的内存区域，每个会话在服务器端都有一块专有的内存区域就是 PGA。本文主要对 SGA 进行分析描述。SGA 组成如图 3-3 所示。

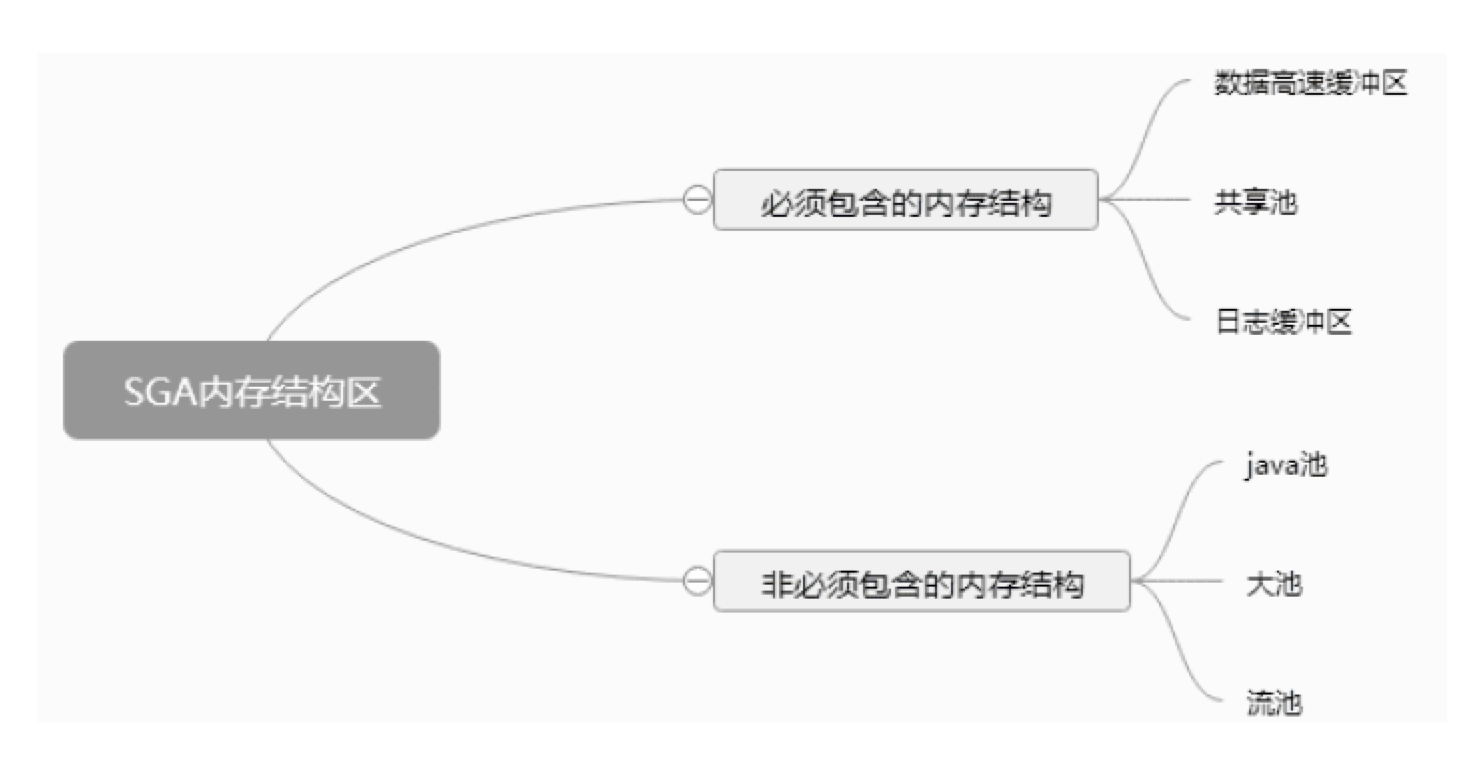


图 3-3　SGA 内存结构

3.3.1　数据库高速缓冲区

缓冲区缓存是 Oracle 用来执行 SQL 语句的工作区域，在更新数据时，用户会话不会直接去更新磁盘上的数据。

【示例 3-1】查询语句

```
select ename,salary from emp where name='SAM';
```

如示例 3-1 这样一条简单的查询语句，Oracle 是如何处理的呢？首先，当用户提交了该条 sql 语句，由对应的用户进程（比如常用的 sql developer）将其发送给服务器，监听程序监听到该条请求，会为其建立一个对应的服务器进程，然后服务器进程会先扫描缓冲区中有没有包含关键行（"SAM"）的数据块，如果有，这就算一次缓存命中了，然后相关行会传输到 PGA 进行进一步处理，最终经过格式化后展示给用户；如果没有命中，那么服务器进程会首先将对应行复制到缓冲区内，然后返回给客户端。

DML（insert，update，delete）操作同理，加入用户发送一条 update 语句，服务进程依然先去扫描缓冲区，如果缓存命中，则直接更新，数据变脏；如果没有命中，由服务器进程将对应数据块先从磁盘上复制到缓冲区内，再进行更新操作。

如果缓冲区存储的块和磁盘上的块不一致，该缓冲区就叫作“脏缓冲区”，脏缓冲区最终会由数据库写入器（DBWn）写入到磁盘中去。

数据库写入器是 Oracle 的一个后台进程。所谓后台进程，是相对于前台进程（服务器进程）来讲的。DBWn 的"n"意味着一个实例是可以有多个数据库写入器的。简而言之，DBWn 的作用就是将变脏了的缓冲区从数据库缓冲区缓存中写入到磁盘中的数据文件中去。

数据库缓冲区缓存区域和数据库写入器是比较重要的概念，别的数据库产品像 MySQL 也都有对应的实现，只不过叫法不一样罢了。了解时，要时刻意识到会话是不会直接更新磁盘数据的，会话的更新、插入、删除包括查询等都是先作用到缓冲区上，随后，DBWn 会将其中的脏缓冲区转储到磁盘上去。

那么 DBWn 什么时候写入呢？DBWn 是一个比较懒的进程，会尽可能少地进行写入，在以下四种情况下会执行写入：

- 没有任何可用缓冲区（不得不写）。
- 脏缓冲区过多。
- 3 秒超时（最晚 3 秒会执行一次写入）。
- 遇到检查点，即 checkPoint（检查点）。检查点是一个 Oracle 事件，遇到检查点，DBWn 会执行写入。比如，实例有序关闭时会有检查点，DBWn 会将所有脏缓冲区写入到磁盘上去，这很容易理解，要保持数据文件的一致性。

从上述 DBWn 的几个写入时机可以看出，DBWn 的写入不直接依赖于会话的更新操作。不是一有脏缓冲区，它就执行写入。而且，DBWn 执行写入跟 commit 操作也没有任何关系，不要以为 commit 操作的影响结果会实时流入到磁盘中去。

DBWn 采用极懒算法进行写入，原因应该要清楚：频繁的磁盘 IO 对系统的压力很大，如果 DBWn 很积极地去写入磁盘，那对系统性能的影响就太大了，换个角度想，如果 DBWn 很勤快地写磁盘，那么数据库缓冲区存在的意义也就不大了。

当然，说到这里，用户可能会意识到一个问题，DBWn 如此懒地进行数据转储，如果在某一时刻，数据库缓冲区缓存内存在着大量的脏缓冲区（在生产环境中，这是常态），也就是有大量的未 commit 和已 commit 的数据还在内存中，没有持久化到磁盘中，然后突然系统断电了，这种情况下，数据是不是就丢掉了？数据当然不会丢失，这就引出了重做日志（redo log）的概念。接下来，就来谈谈对应重做日志的内存结构和后台进程。

3.3.2 日志缓冲区

当用户执行一些 DML 操作（INSERT、UPDATE、DELETE）时，数据块发生改变，产生的变更向量则会写入到重做日志文件中去。有了这些记录，当系统由于断电等因素突然宕掉，数据库缓冲区缓存内的大量脏数据还没来得及写入到数据文件中去，在重新启动时，会有一个实例恢复的过程，在此过程中就应用了重做日志记录来使数据保持一致；或者数据库遭遇了物理损坏，比如磁盘损坏了，此时可以通过 Oracle 的备份恢复工具（如 RMAN）进行数据恢复，原理就是提取备份集然后应用重做日志文件中的变更记录。

日志缓冲区是一块比较小的内存区域，用来短期存储将写入到磁盘中的重做日志文件中的变更向量的。日志缓冲区存在的意义依然是为了减少磁盘 IO，减少用户的等待时间。试想下，如果每一次用户 DML 操作都要进行等待重做记录被写入到磁盘中去，用户体验会比较差。

日志的写入工作是由日志写入器来负责完成的。顾名思义，日志写入器（LGWR）就是把日志缓冲区内的内容写入到磁盘的重做日志文件中去，相比数据库写入器（DBWn），日志写入器就勤快多了。

以下三种情况 LGWR 会执行写入：

（1）Commit 时写入：因为 DBWn 的写入和 commit 没有任何关系，如果 commit 时数据库没有任何记录，那数据就真的丢失了，Oracle 的重做日志就是为了保证数据安全而存在的，commit 时，会话会先挂起，等待 LGWR 将这些记录写入到磁盘上的重做日志文件中，才会通知用户提交完成。所以，LGWR 在 commit 时执行写入，是为了确保事务永不丢失。

（2）日志缓冲区的占用率达到 1/3。

（3）DBWn 要写入脏缓冲区前，这个写入是为了数据回滚考虑的。DBWn 完全可能写入还没提交的事务（参照上面提到的写入时机），那如何保证事务回滚呢？

首先要知道，DBWn 除了写入实际的数据，还会写入撤销数据。简单说，事务回滚需要撤销数据，在写入撤销数据前，会先写入针对撤销数据的日志记录，若用户要进行事务回滚，就可以应用这些日志记录来构造撤销数据，然后进行回滚。

下面对这两块最重要的内存区域和对应的后台进程做个总结：数据库缓冲区缓存和日志缓冲区都是为了提高性能，避免频繁 IO 而存在的。日志缓冲区相比数据库缓冲区缓存要小得多，并且不能进行自动管理，对于日志缓冲区的修改需要重启实例，数据库缓冲区缓存可进行自动管理。作用在数据库缓冲区缓存上的 DBWn 进程，为了避免频繁的磁盘 IO 导致系统性能下降，会尽可能少地执行写入，且 DBWn 的写入和 commit 操作没有任何关系。作用在日志缓冲区上的 LGWR 进程，则会非常积极地进行写入，一般情况下，它几乎是实时地将重做日志记录转储到磁盘中去。LGWR 是 Oracle 体系结构中最大的瓶颈之一。DML 的速度不可能超过 LGWR 将变更向量写入磁盘的速度。

下面继续讲解其他的内存区域和后台进程。

3.3.3　共享池

共享池是最复杂的 SGA 结构，它有许多子结构，下面来看看常见的几个共享池组件。

（1）库缓存：库缓存这块内存区域会按已分析的格式缓存最近执行的代码。这样，同样的 sql 代码多次执行时，就不用重复地去进行代码分析，可以在很大程度上提高系统性能。

（2）数据字典缓存：存储 Oracle 中的对象定义（表、视图、同义词、索引等数据库对象）。这样在分析 sql 代码时，就不用频繁去磁盘上读取数据字典中的数据了。

（3）PL/SQL 区：缓存存储过程、函数、触发器等数据库对象，这些对象都存储在数据字典中，通过将其缓存到内存中，可以在重复调用时提高性能。

（4）大池：大池是一个可选的内存区域。前面提到专有服务器连接和共享服务器连接，如果数据库采用了共享服务器连接模式，则要使用到大池；RMAN（Oracle 的高级备份恢复工具）备份数据也需要大池。

3.3.4　Java 池和流池

Java 池和流池因为实际的企业环境中使用的比较少，这里不做过多的介绍。Oracle 的很多选项是使用 Java 写的，Java 池用作实例化 Java 对象所需的堆空间。从重做日志中提取变更记录的进程和应用变更记录的进程会用到流池（如实例不正常关闭，譬如断电导致实例关闭，在重启时，Oracle 会自动执行实例恢复过程，在此过程需要提取重做日志记录和应用重做日志两个动作）。

3.4 后台进程

下面本章将介绍有关于 Oracle 数据库的进程相关知识，重点介绍与日常管理工作紧密相关的后台进程。

Oracle 进程又分为两类：服务器进程和后台进程。

服务器进程用于处理连接到该实例的用户进程的请求。当应用和 Oracle 是在同一台机器上运行，而不再通过网络，一般会将用户进程和它相应的服务器进程组合成单个的进程，可降低系统开销。当应用和 Oracle 运行在不同的机器上时，用户进程经过一个分离服务器进程与 Oracle 通信。它可执行下列任务：

（1）对应用所发出的 SQL 语句进行语法分析和执行。

（2）从磁盘（数据文件）中读入必要的数据块到 SGA 的共享数据库缓冲区（该块不在缓冲区时）。

（3）将结果返回给应用程序处理。

系统为了使性能最好和协调多个用户，在多进程系统中使用一些附加进程，称为后台进程。在许多操作系统中，后台进程是在实例启动时自动建立的。一个 Oracle 实例可以有许多后台进程，但它们不是一直存在。

数据库实例有内存结构和后台进程。应用与数据库的所有操作和交互都由数据库实例完成，SGA 可以理解为交互平台，后台进程则可以理解为 SGA 与数据库交互的桥梁。PMON、SMON、DBWRn、LGWRn、CKPT 进程为必需的后台进程，ARCHn、LCKn 等为可选后台进程，如图 3-4 所示。

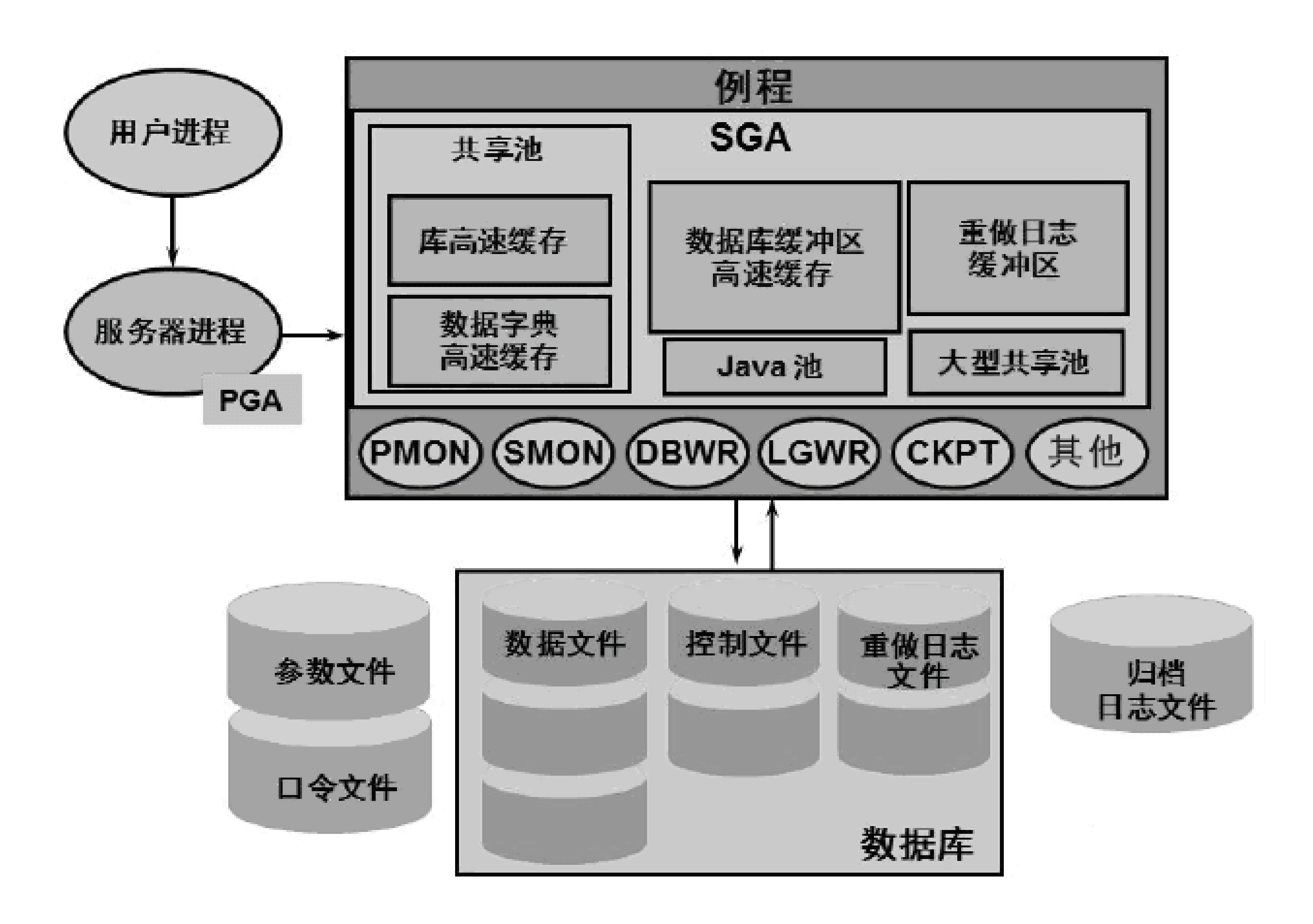


图 3-4 数据库后台进程机构

【示例 3-2】Oracle 数据库各个进程的启动顺序

```
PMON started with pid=2, OS id=18042
```

```
DIAG started with pid=3, OS id=18044
PSP0 started with pid=4, OS id=18051
LMON started with pid=5, OS id=18053
LMD0 started with pid=6, OS id=18055
LMS0 started with pid=7, OS id=18057
MMAN started with pid=8, OS id=18061
DBW0 started with pid=9, OS id=18063
LGWR started with pid=10, OS id=18065
CKPT started with pid=11, OS id=18067
SMON started with pid=12, OS id=18069
RECO started with pid=13, OS id=18071
CJQ0 started with pid=14, OS id=18073
MMON started with pid=15, OS id=18075
MMNL started with pid=16, OS id=18077
```

每个后台进程与 Oracle 数据库的不同部分交互。其中，SMON、PMON、DBWn、CKPT、LGWR 是五个必需的 Oracle 后台进程。

3.4.1　进程监视器

进程监视器（Process Monitor，PMON）主要监视服务器进程。在专有服务器体系模式下，用户进程和服务器进程是一对一的关系，如果某个会话发生异常，PMON 会销毁对应的服务器进程、回滚未提交的事务并回收会话专有的 PGA 内存区域。例如，如果因某些原因专用服务“故障”或被“杀死”，PMON 就负责处理（恢复或回滚工作）和释放资源。PMON 将发出未提交工作的回滚、释放锁和释放分配给故障进程的 SGA 资源。

除了在异常中断之后的清理外，PMON 监控其他 Oracle 后台进程，如果有必要（和有可能）就重新启动。如果共享服务或一个分配器故障（崩溃），PMON 将插手并且重启另一个（在清理故障进程之后）。PMON 将观察所有 Oracle 进程，只要合适或重启它们或中止进程。例如，在数据库日志写进程事件中的 LGWR 故障、实例故障。这是一个严重的错误，最安全的处理方法就是去立即终止实例，让正常的恢复处理数据。



这是很少发生的事情，应该立即报告 Oracle 支持。

PMON 为实例做的另一件事是去使用 Oracle TNS 监听器登记。当一个实例开启时，PMON 进程投出众所周知的端口地址，除非指向其他，来看监听器是否正在开和运行着。众所周知，默认端口是使用 1521。现在，如果监听器在一些不同端口开启会发生什么？这种情况下，机制是相同的，除了监听器地址需要被 LOCAL_LISTENER 参数明确指定。如果监听器运行在库实例开启时，PMON 和监听器通信，传到它相关参数，譬如服务器名和实例的负载度量。如果监听器没被开启，PMON 将周期性地试着和它联系来登记自己。

3.4.2　系统监视器

系统监控后台进程（System Monitor Process，SMON）有时也被叫作 system cleanup process，这么叫的原因是它负责完成很多清理（cleanup）任务。安装和打开数据库，实例恢复是由此

进程完成的。

这个进程对于 Oracle 数据库来说，可以利用一句话来概括，即人小鬼大。其负责的内容并不是很多，但是对于数据的安全与数据库的性能却有很关键的作用。如随着表空间中的数据不断的建立、删除、更新等，在表空间中难免会产生碎片。由于这些碎片的存在，数据库的性能会逐渐降低，而系统监视进程 SMON 正好可以解决这些碎片。SMON 进程会将各个表空间的空闲碎片合并在一起，让数据库系统更加容易分配，从而提高数据库的性能。

另外，在数据库运行的过程中，会因为断电或者其他的原因而发生故障。此时由于数据高速缓存中的脏缓存块还没有来得及写入到数据文件中，从而导致数据的丢失。在数据库启动时，系统监视进程 SMON 会在下一次启动例程时自动读取重做日志文件并对数据库进行恢复。也就是说，进行将已提交的事务写入数据文件（已经写入到日志文件中而没有写入到数据文件中的数据）、回退未提交的事务操作。可见，SMON 进程在 Oracle 数据库中是一个比较小但是却非常重要的角色。

在管理这个进程时，主要需要注意两个问题。

（1）启动的时机。一般情况下，例程重新启动时，会启动这个系统监视进程。然后在这个例程运行期间，这个进程也会被系统定期唤醒，然后会检查是否有工作需要其完成。最重要的是，在有需要时，数据库管理员可以通过其他进程来启动这个 SMON 系统监视进程来完成一些特定的工作。

（2）表空间配置对这个进程的影响。在表空间管理中，有一个参数叫作 PCTINCREASE。如果将这个参数设置为 0 的话，则这个 SMON 系统监视进程对于这个表空间的作用就要打折扣了。在设置为 0 的情况下，SMON 进程就不会对这个表空间中的空闲碎片进行整理、合并操作。也就是说，需要数据库管理员通过数据的导出导入等手工操作才能够解决表空间的碎片问题。显然这会增加数据库管理员的工作量。为此建议，除非有特别的需要，不要将这个参数设置为 0。让 SMON 进程自动对表空间中的碎片进行管理，自动合并表空间中的空闲碎片。不将某个表空间中的这个参数设置为 0 的话，也不会影响到系统监视进程的其他用途，如不会影响到在例程非正常关闭时对数据的恢复操作。即这个参数设置为 0 ，在有需要时仍然可以利用重做日志文件中的记录来恢复相关的数据。

3.4.3 检查点管理进程

检查点管理进程（Checkpoint Process，CKPT）负责发起检查点信号。

【示例 3-3】手动设置检查点

```
SQL>alter system checkpoint;
```

检查点可强制 DBWn 写入脏缓冲区，当数据库崩溃后，由于大量脏缓冲区未写入数据文件，在重新启动时，需要由 SMON 进行实例恢复，实例恢复需要提取和应用重做日志记录，提取的位置就是从上次检查点发起的位置开始的(检查点之前的数据已经被强制写入到数据文件中），这个位置称为 RBA（Redo Byte Address）。CKPT 会不断将这个位置更新到控制文件

中去（以确定实例恢复需要从哪儿开始提取日志记录）。

3.4.4　数据库写进程

上文在讲解 Oracle 内存结构时，已经就数据库进程（Database Writer，DBWn）的作用做了介绍。该进程负责将脏数据块写入磁盘。它是一个非常重要的进程，随着内存的不断增加，一个 DBWn 进程可能不够用了。所以从 Oracle 8i 起就可以为系统配置多个 DBWn 进程。初始化参数 db_writer_processe 决定了启动多少个 DBWn 进程。每个 DBWn 进程都会分配一个 cache buffers lru chain latch。

DBWn 作为一个后台进程，只有在某些条件满足了才会触发。这些条件包括：

- 当进程在 LRU 链表扫描以查找可以覆盖的 buffer header 时，如果已经扫描的 buffer header 的数量到达一定的限度时，触发 DBWn 进程。
- 如果脏数据块的总数超过一定限度，也将触发 DBWn 进程。
- 发生检查点（包括增量检查点（Incremental Checkpoint）和完全检查点（Complete Checkpoint））时触发 DBWn。
- 每隔三秒钟启动一次 DBWn。

3.4.5　日志写进程

日志写进程（Log Writer，LGWR）也是一种后台进程，主要负责将日志缓冲内容写到磁盘的在线重做日志文件或组中。DBWn 将 dirty 块写到磁盘之前，所有与 buffer 修改相关的 redo log 都需要由 LGWR 写入磁盘的在线重做日志文件（组）。如果未写完，那么 DBWn 会等待 LGWR，也会产生一些相应的等待事件（例如，log file prarllel write，后面单独作为话题再聊）。总之，这样做的目的就是为了当 crash 时，可以有恢复之前操作的可能，也是 Oracle 在保持交易完整性方面的一个机制。

该进程有如下几方面的特点：

（1）LGWR 写日志是顺序写，这就解释了一个 Orace Server 只能有一个 LGWR 进程，不能像 DBWR 那样可以有多个，否则就无法保证顺序写的机制，而且可能会产生锁的问题。

（2）用户进程每次修改内存数据块时，都会在日志缓冲区（redo buffer）中构造一个相应的重做条目（redo entry），它记录了被修改数据块修改之前和之后的值。

（3）LGWR 将 redo entry 写入联机日志文件的情况可以概括为两种：后台写和同步写，或者说异步写和同步写。

① 后台写的条件如下：

- 每 3 秒 LGWR 启动一次。
- DBWR 启动时如果发现 dirty 块对应的 redo entry 还没写入联机日志文件，则 DBWR 触发 LGWR 进程并等待 LGWR 完成后继续。
- redo entry 数量达到整个 log buffer 的 1/3 时，触发 LGWR。

- redo entry 的数量达到 1MB。

② 同步写的条件是：执行 commit 时，必须等待 log buffer 进行 flushing 操作（可能产生 log file sync 等待事件），写入磁盘中的联机日志文件。一般上述 1/3 满的条件触发 LGWR，几乎强制 LGWR 实时写，因此当需要执行 commit 时，可能没有任何 redo entry 需要写入了。

（4）3 秒触发 LGWR 的规则，事实上，这个超时是 DBWR 的，但是因为 LGWR 总在 DBWR 调用之前执行，因此效果上也相当于 LGWR 的超时是 3 秒即调用。

3.4.6 管理监控进程

管理监控进程（Manageability Monitor，MMON）是数据库的自我监视和自我调整的支持进程。实例在运行中，会收集大量有关实例活动和性能的统计数据，这些数据会收集到 SGA 中，MMON 定期从 SGA 中捕获这些统计数据，并将其写入到数据字典中，便于后续对这些快照进行分析。（默认情况下，MMON 每隔一个小时收集一次快照）。

3.4.7 归档进程

归档进程（Archiver，ARCn）是可选的。在重做日志文件管理中，有归档与非归档两种模式。如果数据库配置为归档模式，这个进程就是必需的。所谓归档，就是将重做日志文件永久保存（生产库一般都会配置为归档模式）到归档日志文件中。归档日志文件和重做日志文件的作用是一样的，只不过重做日志文件会不断被重写，而归档日志文件则保留了关于数据更改的完整的历史记录。

在日志进行切换时，如果不对原先的日志文件进行归档，而直接覆盖的话，就叫作非归档模式。相反，在写入下一个日志文件时，会先对目标日志文件进行归档，这就叫作归档模式。归档进程 ARCH 就是负责在重做日志文件切换后将已经写满的重做日志文件复制到归档日志文件中，以防止循环写入重做日志文件时将其覆盖。

所以说，只有数据库运行在归档模式时，这个 ARCH 进程才会被启动。在任何一种操作模式下，重做日志文件都会被循环使用。所以当 LGWR 进程在进行日志切换，需要用到下一个日志文件时，数据库会被暂时挂起，进行目标日志文件的归档工作。直到这个目标重做日志文件归档完毕后，数据库才会恢复正常。所以说，归档日志的操作有时候也会影响数据库的性能，特别是当需要进行频繁的大批量数据更改时。

那么有什么方法可以提高归档作业的效率呢？如下一些建议可供数据库管理员参考。

（1）增加归档进程的个数。

在默认情况下，一个例程只会启动一个归档进程 ARCH。当 ARCH 进程正在归档一个重做日志文件时，任何其他的进程都不能够访问这个重做日志文件。在 Oracle 数据库中，可以根据需要启动多个归档进程 ARCH。在 Oracle 数据库中，启动多个归档进程时分为手工与自动两个方式。为了提高重做日志文件归档的速度，当用户进程发生比较长时间的等待时，LGWR 进程会根据时机情况来自动启动多个归档进程。在 Oracle 数据库中，其最多可以启动十个归

档进程。另外，如果数据库管理员在部署数据库时，估计日志归档作业会影响到数据库的性能，就可以手工来启动多个归档进程。这是通过初始化参数 LOG_ARCHIVE_MAX_PROCESSES 来确定的。可以将这个参数设置为大于 1 的数值（注意不能够超过 9 个归档进程）。如此的话，数据库在创建例程时就会启动多个归档进程。不过还是倾向于让数据库系统来自动管理这个进程。数据库管理员最好不要干涉。

另外，需要注意 ARCH 归档进程个数与 DBWR 进程个数的区别。默认情况下，DBWR 进程也只有一个。为了提高数据库的性能，可以根据情况增加 DBWR 进程的个数。不过其增加时受到 CPU 数量的限制，即一个 DBWR 进程需要使用一个独立的 CPU。如果想启动三个 DBWR 进程的话，就必须采用 3 个 CPU 处理器。对于 ARCH 归档进程来说，则没有这个限制。即使只有一个 CPU 处理器，其也可以启动三个甚至更多的 ARCH 进程。

（2）增加重做日志文件来延长归档日志进程启动的时间间隔。

通常情况下，只有当前一个重做日志文件写满、需要进行日志切换时，才会触发这个 ARCH 归档日志进程。所以如果重做文件比较大，其日志切换的时间间隔就会延长，ARCH 归档日志进程的启动时间间隔也会比较长。所以说，通过调整重做日志文件的大小，可以延长归档进程启动的时间间隔，从而降低因为归档进程启动而对数据库性能造成的负面影响。

（3）在数据库初始化的过程中，可能需要导入大量的数据。

此时会对数据库中的数据进行大量的插入、删除、更新等操作，从而导致重做日志文件切换频繁。这就会导致数据库需要频繁启动 ARCH 归档进程。

数据库大量的更新操作、重做日志文件（LGWR 进程）、归档重做日志文件（ARCH）进程之间就形成了一条无形的链条。由于“蝴蝶效应”，从而降低了数据库的性能。为此在必要时，需要砍断这根链条，以提高数据库的性能。例如，可以在数据大量导入、更新、删除时，不往日志文件中插入记录，或者临时增加重做日志文件的空间。如此的话，在进行这些操作时就可以避免进行重做日志切换或者延长重做日志切换的时间间隔，从而使 ARCH 归档日志进程也可以避免或者延长其时间间隔，从而提高数据库的性能。当数据库初始化完成之后，再将其恢复过来。这些临时性的调整虽然比较麻烦，但是可以提高数据库的性能。为此认为这是值得的。

Oracle 是如何保存数据的呢？众所周知，内存的数据处理速度是比物理磁盘快很多的，Oracle 数据库的所有数据更改都在内存中完成，当然，在某些情况下，某些读取类的操作是不会经过 SGA 内存区域的，而是选择直接从硬盘将数据获取并返回给用户。在实际的数据库管理工作中，管理员更多的工作是与内存中的数据相关。数据库是存放数据的容器，那么数据是如何放进这个容器中的呢？内存数据块写入数据文件其实是一个相当复杂的过程，在这个过程中，首先要保证安全。所谓安全，就是在写的过程中，一旦发生实例崩溃，要有一套完整的机制能够保证用户已经提交的数据不会丢失；其次，在保证安全的基础上，要尽可能地提高效率。众所周知，I/O 操作是最昂贵的操作，所以应该尽可能地将脏数据块收集到一定程度以后，再批量写入磁盘中。

直观上最简单的解决方法就是，每当用户提交时就将所改变的内存数据块交给 DBWn，由其写入数据文件。这样的话，一定能够保证提交的数据不会丢失，但是这种方式效率最为低

下，在高并发环境中，一定会引起 I/O 方面的争用。Oracle 当然不会采用这种没有伸缩性的方式。Oracle 引入了 CKPT 和 LGWR 这两个后台进程，这两个进程与 DBWn 进程互相合作，提供了既安全又高效的写脏数据块的解决方法。

用户进程每次修改内存数据块时，都会在日志缓冲区（log buffer）中构造一个相应的重做条目（redo entry），该重做条目描述了被修改的数据块在修改之前和修改之后的值，而 LGWR 进程则负责将这些重做条目写入联机日志文件。只要重做条目进入了联机日志文件，那么数据的安全就有保障了，否则这些数据都是有安全隐患的。LGWR 是一个必须和前台用户进程通信的进程。LGWR 承担了维护系统数据完整性的任务，它保证了数据在任何情况下都不会丢失。

假如 DBWR 在写脏数据块的过程中突然发生实例崩溃该怎么办呢？用户提交时，Oracle 是不一定会把提交的数据块写入数据文件的。那么实例崩溃时，必然会有一些已经提交但是还没有被写入数据文件的内存数据块丢失了。当实例再次启动时，Oracle 需要利用日志文件中记录的重做条目在 buffer cache 中重新构造出被丢失的数据块，从而完成前滚和回滚的工作，并将丢失的数据块找回来。于是这里就存在一个问题，就是 Oracle 在日志文件中找重做条目时，到底应该找哪些重做条目？换句话说，应该在日志文件中从哪个起点开始往后应用重做条目？注意，这里所指的日志文件可能不止一个日志文件。

因为需要预防随时可能的实例崩溃现象，所以 Oracle 在数据库的正常运行过程中会不断地定位这个起点，以便在不可预期的实例崩溃中能够最有效地保护并恢复数据。同时，这个起点的选择非常有讲究。首先，这个起点不能太靠近日志文件的头部，太靠近日志文件头部意味着要处理很多的重做条目，这样会导致实例再次启动时所进行恢复的时间太长；其次，这个起点也不能太靠近日志文件的尾部，太靠近日志文件的尾部说明只有很少的脏数据块没有被写入数据文件，也就是说前面已经有很多脏数据块被写入了数据文件，那也就意味着只有在 DBWn 进程很频繁地写数据文件的情况下，才能使得 buffer cache 中所残留的脏数据块的数量很少。但很明显，DBWn 写得越频繁，所占用写数据文件的 I/O 就越严重，那么留给其他操作（比如读取 buffer cache 中不存在的数据块等）的 I/O 资源就越少。这显然也是不合理的。

从这里也可以看出，这个起点实际上说明了在日志文件中位于这个起点之前的重做条目所对应的在 buffer cache 中的脏数据块已经被写入了数据文件，从而在实例崩溃以后的恢复中不需要去考虑，而这个起点以后的重做条目所对应的脏数据块实际还没有被写入数据文件。在实例崩溃以后的恢复中，需要从这个起点开始往后依次取出日志文件中的重做条目进行恢复。考虑到目前的内存容量越来越大，buffer cache 也越来越大，buffer cache 中包含几百万个内存数据块也是很正常的现象，如何才能最有效地来定位这个起点呢？

为了能够确定这个最佳的起点，Oracle 引入了名为 CKPT 的后台进程，通常也叫作检查点进程（Checkpoint Process）。这个进程与 DBWn 共同合作，从而确定这个起点。同时，这个起点也有一个专门的名字，叫作检查点位置（Checkpoint Position，该检查点位置记录在控制文件里）。

Oracle 为了在检查点的算法上更加具有可扩展性（也就是为了能够在巨大的 buffer cache 下依然有效工作），引入了检查点队列（Checkpoint Queue），该队列上串起来的都是脏数据块所对应的 buffer header。每次 DBWn 写脏数据块时，也是从检查点队列上扫描脏数据块，并将这些脏数据块实际写入数据文件的。当写完以后，DBWn 会将这些已经写入数据文件的脏

数据块从检查点队列上摘下来。

这样即便是在巨大的 buffer cache 下工作，CKPT 也能够快速地确定哪些脏数据块已经被写入了数据文件，而哪些还没有写入数据文件。显然，只要在检查点队列上的数据块就都是还没有写入数据文件的脏数据块。而且，为了更加有效地处理单实例和多实例（RAC）环境下的表空间的检查点处理，比如将表空间设置为离线状态或者为热备份状态等，Oracle 还专门引入了文件队列（File Queue）。文件队列的原理与检查点队列是一样的，只不过每个数据文件会有一个文件队列，该数据文件所对应的脏数据块会被串在同一个文件队列上；同时为了能够尽量减少实例崩溃后恢复的时间，Oracle 还引入了增量检查点（Incremental Checkpoint），从而增加了检查点启动的次数。

如果每次检查点启动的间隔时间过长的话，再加上内存很大，可能会使得恢复的时间过长。因为前一次检查点启动以后，标识出了这个起点。然后在第二次检查点启动之前，DBWn 可能已经将很多脏数据块写入了数据文件，而假如在第二次检查点启动之前发生实例崩溃，导致在日志文件中所标识的起点仍然是上一次检查点启动时所标识的，导致 Oracle 不知道这个起点以后的很多重做条目所对应的脏数据块实际上已经写入了数据文件，从而使得 Oracle 在实例恢复时重复地处理一遍，效率低下，浪费时间。

上面说到了有关 CKPT 的两个重要概念：检查点队列（包括文件队列）和增量检查点。检查点队列上的 buffer header 是按照数据块第一次被修改的时间先后顺序来排列的。越早修改的数据块的 buffer header 排在越前面，同时如果一个数据块被修改了多次的话，在该链表上也只出现一次。而且，检查点队列上的 buffer header 还记录了脏数据块在第一次被修改时所对应的重做条目在重做日志文件中的地址，也就是 LRBA（Low Redo Block Address）。Low 表示第一次修改时对应的 RBA。每个检查点都会由 checkpoint queue latch 来保护。

而增量检查点是从 Oracle 8i 开始出现的，是相对于 Oracle 8i 之前的完全检查点（Complete Checkpoint）而言的。完全检查点启动时，会标识出 buffer cache 中所有的脏数据块，然后以最高优先级启动 DBWn 进程将这些脏数据块写入数据文件。Oracle 8i 之前，日志切换时会触发完全检查点。到了 Oracle 8i 及以后，完全检查点只有在两种情况下才会被触发：

- 发出 alter system checkpoint 命令。
- 除了 shutdown abort 以外的正常关闭数据库。



日志切换不会触发完全检查点，而是触发增量检查点。自 Oracle 8i 所引入的增量检查点每隔三秒钟或发生日志切换时启动。它启动时只做一件事情：找出当前检查点队列上的第一个 buffer header，并将该 buffer header 中所记录的 LRBA（这个 LRBA 也就是 checkpoint position）记录到控制文件中去。如果是由日志切换所引起的增量检查点，则还会将 checkpoint position 记录到每个数据文件头中。也就是说，如果这个时候发生实例崩溃，Oracle 在下次启动时就会到控制文件中找到这个 checkpoint position 作为日志文件的起点，然后从这个起点开始向后依次取出每个重做条目进行处理。

上面所描述的概念，用一句话来概括，其实就是 DBWn 负责写检查点队列上的脏数据块，而 CKPT 负责记录当前检查点队列的第一个数据块所对应的重做条目在日志文件中的地址，而到底应该写哪些脏数据块、写多少脏数据块则要到检查点队列上才能确定。

3.5 物理结构

Oracle 物理结构包含了数据文件、重做日志文件、控制文件、参数文件、密码文件、归档日志文件、备份文件、告警日志文件、跟踪文件等。其中，数据文件、控制文件、重做日志文件和参数文件是必需的，其他文件可选。

3.5.1 数据文件

每一个 Oracle 数据库都有一个或多个物理的数据文件（Data File）。数据文件包含全部数据库数据，逻辑数据库结构（如表、索引、视图、函数）的数据物理地存储在数据库的数据文件中。数据文件中的数据在需要时可以读取并存储在 Oracle 内存储区中。例如，用户要存取数据库一表的某些数据，若请求信息不在数据库的内存存储区内，则从相应的数据文件中读取并存储在内存，当修改或插入新数据时，为了减少磁盘输出的总数、提高性能，不必立刻写入数据文件，数据存储在内存，然后由 Oracle 后台进程 DBWRn 决定如何将其写入到相应的数据文件。

数据文件有下列特征：

一个数据文件仅与一个数据库联系。

一个表空间（数据库存储的逻辑单位）由一个或多个数据文件组成。

3.5.2 日志文件

每一个数据库实例有两组或以上日志文件组（Redo Log Files），为了防止日志文件本身的故障，每个日志文件组可以有一个或以上日志成员。日志的主要功能是记录对数据所做的修改。在出现故障时，如果不能将修改数据永久地写入数据文件，则可利用日志得到该修改，从而保证数据不丢失。

日志文件中的信息仅在系统故障或介质故障恢复数据库时使用。对于任何丢失的数据，在下一次数据库打开时，Oracle 都会自动地应用日志文件中的信息来恢复数据库数据文件。Oracle 日志文件有联机日志文件和归档日志文件两种。联机日志文件用来循环记录数据库改变的操作系统文件。归档日志文件是为避免联机日志文件重写时丢失重复数据而对联机日志文件所做的备份。Oracle 数据库可以选择归档（ARCHIVELOG）或非归档（NOARCHIVELOG）模式。

3.5.3　控制文件

每一个 Oracle 数据库都有一个控制文件（Control File）或同一个控制文件的多个备份，它记录数据库的物理结构信息，包括数据库名、数据库数据文件和日志文件的名字和位置、数据库建立日期等。由于控制文件记录数据库的物理结构信息，对数据库运行至关重要，为了安全起见，Oracle 建议保存两份以上的控制文件镜像于不同的存储设备。

当 Oracle 数据库的实例启动时，它的控制文件用于标识数据库和日志文件。当着手数据库操作时它们必须被打开。当数据库的物理组成更改时，Oracle 自动更改该数据库的控制文件。当然，在数据恢复时，自然会使用控制文件以确定数据库物理文件的名字和位置。

3.5.4　参数文件

除了构成Oracle数据库物理结构的三类主要文件外，参数文件(Parameter Files)也是Oracle数据库较为重要的一种文件结构。参数文件记录了 Oracle 数据库的基本参数信息，主要包括数据库名、控制文件所在路径、进程等。在 Oracle 9i 之前，都只有 pfile 一种文本格式的参数文件。在 9i 之后，新增了服务器二进制参数文件 SPFILE。通过修改 pfile 以修改数据库参数，必须要求重启数据库后才能生效。通过修改 SPFILE 以修改数据库参数时，根据参数类型分为静态参数需要重启和动态参数无须重启立即生效，可以通过查询 v$parameter 视图确定参数类型。有多种参数文件类型存在，而 Oracle 的正常运行只使用一种参数文件，在 Oracle 启动过程中加载文件的顺序为 spfilesid.ora> spfile.ora> initsid.ora。

3.5.5　其他文件

Oracle 数据的运行除了以上重要的必需文件以外，还有虽然非必需但一样重要的其他文件（Other Files）结构，比如密码文件、归档日志文件、alter 告警日志文件、trace 跟踪文件等。

（1）密码文件的作用是主要进行 DBA 权限的身份认证，用于记录数据库账户管理员的密码，每个数据库必须要拥有密码文件。

（2）归档日志文件只有在数据库开启了归档模式时才会产生，作用是 redo log 的归档备份。

（3）alter 告警日志文件主要记录数据库行为，其中比较重要的信息是里面记录的告警或报错信息，这些对于数据库日常的优化和故障诊断是非常有帮助的。

（4）trace 跟踪文件比 alter 告警日志文件内容更加详细，有一些数据库报错信息会直接写入到 trace 文件中，在 alter 日志中只显示 trace 文件名，供 DBA 深入分析问题。

3.6　逻辑结构

Oracle 数据库的逻辑结构是一种层次结构，主要由表空间、段、区和数据块等概念组成。

逻辑结构是面向用户的，即用户利用 Oracle 开发应用程序使用的就是逻辑结构。数据库存储层次结构及其构成关系，结构对象也从数据块到表空间形成了不同层次的粒度关系，如图 3-5 所示。

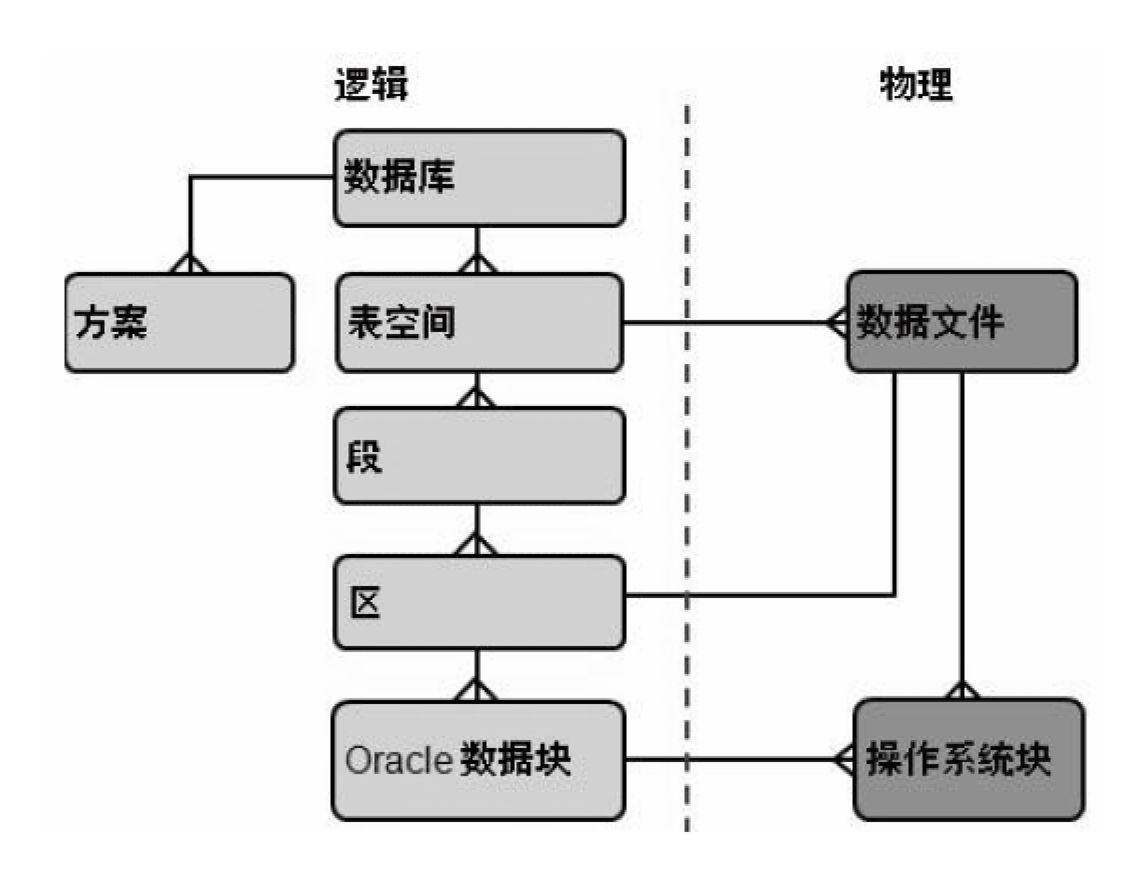


图 3-5　数据库逻辑结构

3.6.1　表空间

表空间（tablespace）是数据库的逻辑划分，任何数据库逻辑对象在存储时都必须存储在某个表空间中。从图 3-5 可以知道每个表空间由多个段组成。表空间一般是由一个或多个数据文件构成的，每个数据文件只能属于某一个表空间，也就是说表空间和数据文件是 1 对 N 的关系。每个数据库至少有一个表空间（system tablespace），表空间大小等于从属于它的所有数据文件大小的总和。在 Oracle 10g 中初始创建的只有 5 个表空间，即 system、temp、undotbs1、sysaux、users。

1. 系统表空间（system tablespace）

system 表空间是每个 Oracle 数据库都必须具备的，数据库创建时自动创建，用于存储数据库系统对象、数据字典、存储过程、触发器和系统回滚段及数据库管理所需的信息；系统表空间的名称是不可更改的，系统表空间必须在任何时候都可以用，也是数据库运行的必要条件。因此，系统表空间是不能脱机的。为避免系统表空间产生存储碎片以及争用系统资源的问题，建议创建独立的用户表空间来单独存储用户对象及数据。

2. 临时表空间（temp tablespace）

temp 表空间相对于其他表空间而言，临时表空间主要用于存储 Oracle 数据库运行期间所产生的临时数据，比如 SQL 排序等。数据库可以建立多个临时表空间。当数据库关闭后，临时表空间中所有数据将全部被清除。

3. 回滚表空间（undo tablespace）

回滚表空间是 Oracle 特有的概念，用于保存 Oracle 数据库变化前的记录，在对数据库中

的记录进行 DML 操作时，Oracle 数据库会将变化前的记录副本保存到回滚表空间中，在 rollback、实例恢复（前滚）、一致性读 CR 块的构造时会使用到 undo 信息，同时保证事务读一致性。在 Oracle 8i 中是 rollback tablespace，从 Oracle 9i 开始改为 undo tablespace。其中，temp 是临时表空间，undotbs1 是 undo 回滚表空间。

4. sysaux 表空间

sysaux 表空间是随着数据库的创建而创建的，它充当 system 的辅助表空间，主要存储除数据字典以外的其他对象，如果启用 EM 或 Gird Control，该表空间就用于存放 EM 采集的监控信息。

5. users 表空间

在创建数据库时自动创建的用户（users）表空间，一般用于维护账户使用的表空间。应用程序表空间一般另外根据应用需求创建。

3.6.2　段

段（Segment）是由多个数据区（Extent）构成的，是为特定的数据库对象（如数据段、索引段、回滚段、临时段）分配的一系列数据区。段内包含的数据区可以不连续，而且可以跨越多个数据文件，使用段的目的是用来保存特定对象。Oracle 数据库分为数据段、索引段、回滚段和临时段 4 种类型。

- 数据段：也称为表段，所包含的数据与表和簇相关。当创建一个表时，系统自动创建一个以该表的名字命名的数据段。
- 索引段：包含索引相关信息，创建索引时，系统自动创建一个以该索引的名字命名的索引段。
- 回滚段：包含回滚信息。DML 操作时，Oracle 数据库会将变化前的记录副本保存到回滚段中，在 rollback、实例恢复（前滚）、一致性读 CR 块的构造时会使用到回滚段信息，同时用于保证事务读一致性。创建数据库时，Oracle 会创建默认的回滚段，其管理方式既可以是自动的，也可以是手工的。
- 临时段：是 Oracle 在运行过程中自行创建的段，当一个 SQL 语句需要临时工作区（比如排序）时，由 Oracle 创建临时段，一旦语句执行完毕，临时段就会自动释放。

3.6.3　数据区

数据区（Extent）是一组连续的数据块。当一个表、回滚段或临时段创建或需要附加空间时，系统总是为之分配一个新的数据区。一个数据区不能跨越多个文件，因为它包含连续的数据块。使用区的目的是用来保存特定数据类型的数据，也是表中数据增长的基本单位。在 Oracle 数据库中，分配空间就是以数据区为单位的。一个 Oracle 对象包含至少一个数据区，设置一个表或索引的存储参数包含设置它的数据区大小。

3.6.4 数据块

数据块（Data Blocks）是 Oracle 最小的存储单位。Oracle 数据存放在“块”中。Oracle 每次请求数据时，都是以块为单位，也就是说，Oracle 每次请求的数据是块的整数倍。如果 Oracle 请求的数据量不到一块，也会读取整个块，因为“块”是 Oracle 读写数据的最小单位或者说最基本的单位。

特别需要注意的是，这里的“块”是 Oracle 的“数据块”，不是操作系统的“块”。Oracle 块的标准大小由初始化参数 DB_BLOCK_SIZE 指定，默认标准块大小为 8KB。具有标准大小的块称为标准块（Standard Block），和标准块的大小不同的块叫非标准块（Nonstandard Block）。同一个数据库实例可以同时存在多种不同的块大小，由初始化参数 DB_BLOCK_SIZE 指定一个标准块大小。操作系统每次执行 I/O 时，都是以操作系统的块为单位的。Oracle 每次执行 I/O 时，都是以 Oracle 的块为单位。Oracle 数据块大小一般是操作系统块的整数倍。

3.7 小结

在学习 Oracle 数据库的过程中，体系结构是重中之重，一开始要从宏观上掌握它的物理结构组成、文件组成和内存组成。掌握得越深入越好，在实际工作中，遇到疑难问题时，往往可以归结到体系结构中来解释。体系结构是对一个系统的框架描述，是设计系统的一个宏观工作。自从 Oracle 推出 ASM 功能模块之后，在学习数据库体系结构时，都会增加 ASM 体系结构的学习，这部分的内容将在下一章节做详细的介绍。ASM 是目前 Oracle 广泛使用的存储方式，也是学习 Oracle 数据库的一部分重要内容。

第 4 章

数据库自动存储管理

自动存储管理（Automatic Storage Management，ASM）是 Oracle Database 的一个特性，为数据库管理员提供了一个在所有服务器和存储平台上均一致的简单存储管理接口。作为专门为 Oracle 数据库文件创建的垂直集成文件系统和卷管理器，ASM 提供了直接异步 I/O 的性能以及文件系统的易管理性。ASM 提供了可节省 DBA 时间的功能，以及管理动态数据库环境的灵活性，并且提高了效率。ASM 的主要优点有：

- 简化和自动化了存储管理
- 提高了存储利用率和敏捷性
- 提供可预测的性能、可用性和可伸缩性

本章将向读者介绍 ASM 的基本概念和特点，帮助读者掌握 ASM 的体系结构。

4.1 ASM 综述

ASM 提供了文件系统与卷管理器的纵向集成，如图 4-1 所示，这是一项专门为 Oracle 数据库文件构建的技术。使用 ASM 可以管理单个 SMP 计算机，也可管理集群的多个节点，以便为 Oracle Real Application Clusters（RAC）提供支持。一个 ASM 实例可以同时为许多 Oracle 数据库提供支持。ASM 将 I/O 负载分布在所有可用的资源中来优化性能，不必手动进行 I/O 优化。

ASM 有助于 DBA 管理动态数据库环境，使 DBA 不必关闭数据库，就可以增加数据库的大小来调节存储分配。ASM 通过维护数据的冗余副本来提供容错能力，也可以基于供应商提供的可靠存储机制来构建 ASM。除了使用本地裸设备以外，ASM 还可以使用存储区网络（SAN）环境中提供的虚拟原始卷，或网络连接存储（NAS）文件服务器上的填零文件。通过为各类数据选择所期望的可靠性和性能特性来进行数据管理，而不是人工处理每个文件。

由于使人工管理存储的工作实现了自动化，ASM 功能使 DBA 节省了很多时间，因此提高了他们管理大型数据库及更多数据库的能力和效率。

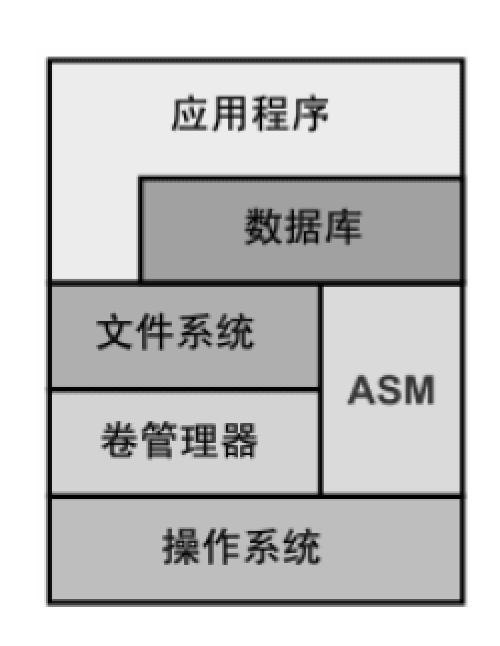


图 4-1　ASM 定位

ASM 将文件分为多个分配单元（AU），并在所有磁盘间平均分配每个文件的分配单元。ASM 使用索引技术跟踪每个分配单元的位置。存储容量发生变化时，ASM 不会重新条带化所有数据，而是根据添加或删除的存储量，按比例移动一定数量的数据，以重新均匀分配文件，保持磁盘间的负载平衡。此操作是在数据库处于活动状态时执行的。可以提高或降低重新平衡操作的速度，以减小对 I/O 子系统的影响。ASM 还提供了镜像保护，因此不必再购买第三方的逻辑卷管理器。ASM 的一个独特优势在于其镜像基于文件，而不是基于卷。因此，同一磁盘组中可以包含受镜像保护的文件和不受镜像保护的文件组合。

ASM 支持数据文件、日志文件、控制文件、归档日志、临时文件、归档日志文件、SPFILE、RMAN 备份集以及其他 Oracle DB 文件类型。ASM 支持 Oracle Real Application Clusters，而且无须使用集群逻辑卷管理器或集群文件系统。

4.2 ASM 体系结构

要使用 ASM，在启动数据库实例之前，必须启动一个名为 ASM 实例的特殊实例，如图 4-2 所示。ASM 实例不会装载数据库，而是管理使 ASM 文件可用于普通数据库实例所必需的那些元数据。ASM 实例和数据库实例都能访问一些公共的磁盘集，这些公共磁盘集称为磁盘组。数据库实例直接访问 ASM 文件的内容，它们与 ASM 实例通信的目的只是为了获取这些文件的布局信息。

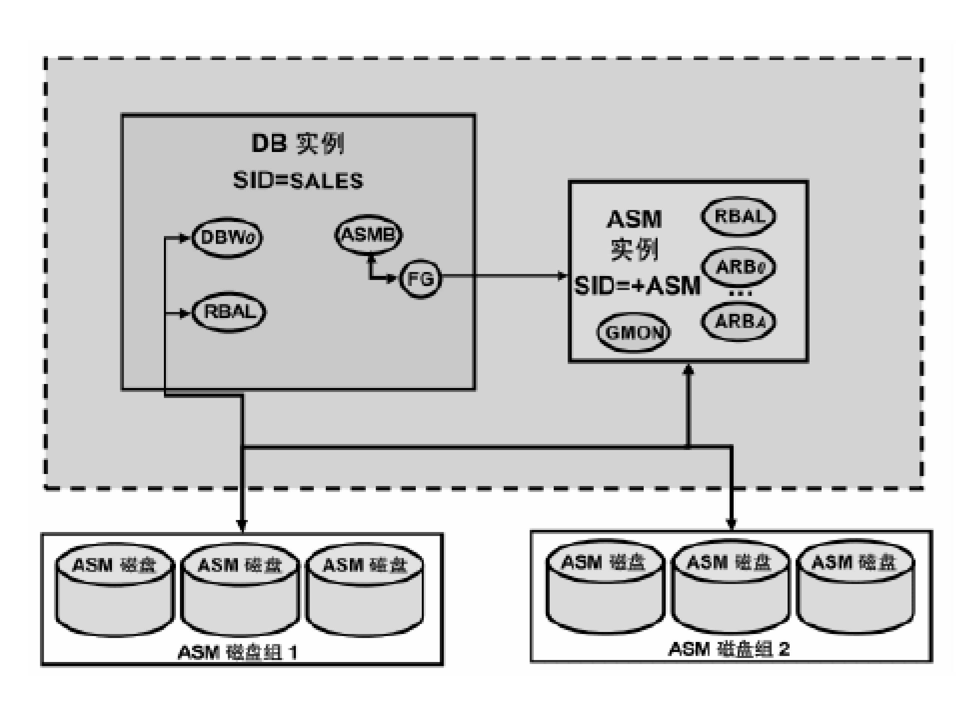


图 4-2　ASM 体系结构

ASM 实例将启动几个特定于 ASM 的后台进程。第一个进程负责协调磁盘组的重新平衡活动，称为 RBAL。第二个进程负责执行实际的重新平衡分配单元移动操作。在同一时刻可以存在许多此类进程，分别命名为 ARB0、ARB1，以此类推。GMON 进程（组监视器）用于合作伙伴和状态表以及节点成员资格。ASM 实例还有一些与数据库实例相同的后台进程，其中包括 SMON、PMON、LGWR、DBWR 和 CKPT。每个使用 ASM 的数据库实例都有两个额外的后台进程 ASMB 和 RBAL。RBAL 负责对磁盘组中的磁盘执行全局打开。数据库实例启动时，ASMB 作为前台进程连接到 ASM 实例。

数据库实例和 ASM 实例之间的通信就是通过这种桥接来实现的。此类通信包括物理文件更改，例如数据文件的创建和删除。通过这种连接可以定期交换消息，以更新统计信息并确认这两个实例都在正常运行。

4.3 ASM 中存储的概念

ASM 不会妨碍任何现有的数据库功能。现有数据库能够像平常一样工作。新文件可以被创建为 ASM 文件，然而可以按原来的方式管理现有文件，也可以将其移植至 ASM。此图描述了 Oracle DB 中不同存储组件之间的关系。图 4-3 的左侧部分和中间部分显示了以前版本中各组件之间的关系。图右侧部分显示了相关的 ASM 概念。但是，这些 ASM 概念仅用于描述文件存储，并没有取代任何现有的概念，如段和表空间。通过 ASM，数据库文件现在可以存储为 ASM 文件。在新层次的顶部是 ASM 磁盘组。

任何单个 ASM 文件只能包含在一个磁盘组中。不过，一个磁盘组中可以包含属于多个数据库的多个文件，并且单个数据库可以使用来自多个磁盘组的存储空间。正如看到的那样，一个磁盘组由多个 ASM 磁盘组成，但每个 ASM 磁盘只能属于一个磁盘组。此外，ASM 文件始终分布在该磁盘组中的所有 ASM 磁盘上。ASM 磁盘按分配单元（AU）进行分区。分配单元是 ASM 分配的最小连续磁盘空间。ASM 不允许跨分配单元拆分物理块。创建磁盘组时，可以按 2 的幂（如 1、2、4、8、16、32 或 64）将 ASM 分配单元大小设置为 1MB 到 64MB 之间的值。注意，此图仅涉及数据文件。不过，ASM 也可用于存储其他类型的数据库文件。

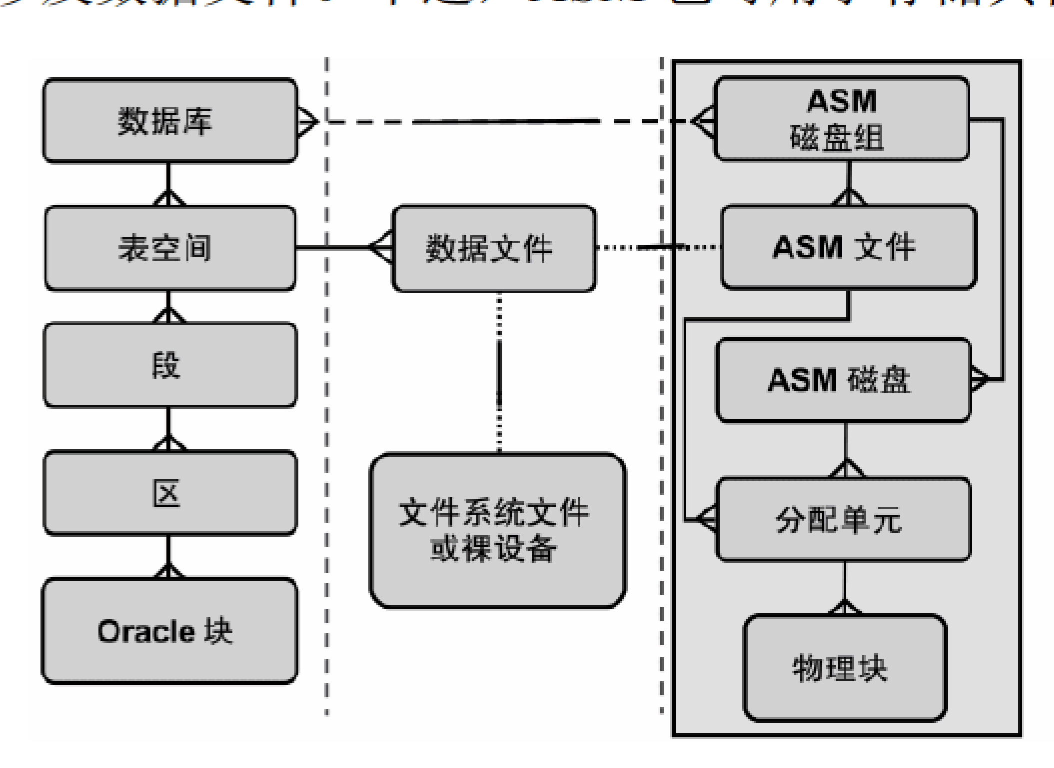


图 4-3　ASM 逻辑划分

虽然在图 4-3 ASM 逻辑划分中的 ASM 逻辑划分中没有体现一种区的概念，但其实这部分是由 ASM 来自动进行管理的，并且是可伸缩的。ASM 可变大小区是一种自动功能，可使用该功能在提高内存使用效率的同时支持较大的文件大小。

ASM 文件的初始区大小等于一个分配单元。随着文件大小的增长，区大小也会按预定义的区数量增加到 8 个分配单元，然后增加到 64 个分配单元。定义文件的区映射大小可以被 8 和 64 整除，具体取决于文件大小。初始区大小等于分配单元大小，随后按预定义的阈值以 8 和 64 为系数成倍增长。描述文件所需的区指针数以及管理共享池中区映射所需的内存都有所减少（在大型文件配置中禁止使用）。

区大小在文件之间以及文件内部都有所不同。通过可变大小区功能还可以使用 ASM 部署几百个太字节（甚至几个拍字节）大小的 Oracle DB。可变大小区的管理是完全自动化的，不需要手动管理。但是，如果分配并释放了大量非连续的小型数据区，并且没有其他连续的大型区可用时，则可能出现外部碎片。碎片整理操作被集成到重新平衡操作中。因此，DBA 始终可以通过执行重新平衡操作对磁盘组进行碎片整理。

4.4 ASM 磁盘组

磁盘组是作为逻辑单元进行管理的一组磁盘。以 ASM 磁盘为单位在磁盘组中添加或删除存储。每个 ASM 磁盘都有一个 ASM 磁盘名，该名称在集群的所有节点中都是相同的。由于不同主机可以使用不同的名称引用同一磁盘，因此必须提供一个 ASM 磁盘名称的缩写。

ASM 磁盘组有一个比较重要的概念：条带化。ASM 条带化有两个主要用途：在磁盘组的所有磁盘之间平衡负载和减少 I/O 等待时间。粗粒度的条带化可以平衡磁盘组的负载，而细粒度的条带化通过更大范围地分配负载，可减少特定文件类型的等待时间。为了使数据条带化，ASM 将文件分成条带，在磁盘组的所有磁盘之间均匀地分布数据。条带的大小等于有效的分配单元大小。粗粒度的条带大小始终等于分配单元大小。细粒度的条带大小始终等于 128KB。这可减少小型 I/O 操作（如重做日志写入）的 I/O 等待时间。

简单总结如下关于 ASM 磁盘组的几个特点：

- 作为逻辑单元管理的磁盘组。
- 将磁盘总空间划分为统一大小的单元。
- 将各个文件均匀分配到所有磁盘中。
- 根据文件类型使用粗粒度或细粒度的条带化。
- 管理的是磁盘组而非文件。

上面介绍了磁盘组的相关概念，除此之外，在 ASM 中还存在故障组和磁盘镜像的一些知识，对于数据库的日常维护同样具有重要的作用，比如故障组，如图 4-4 所示。

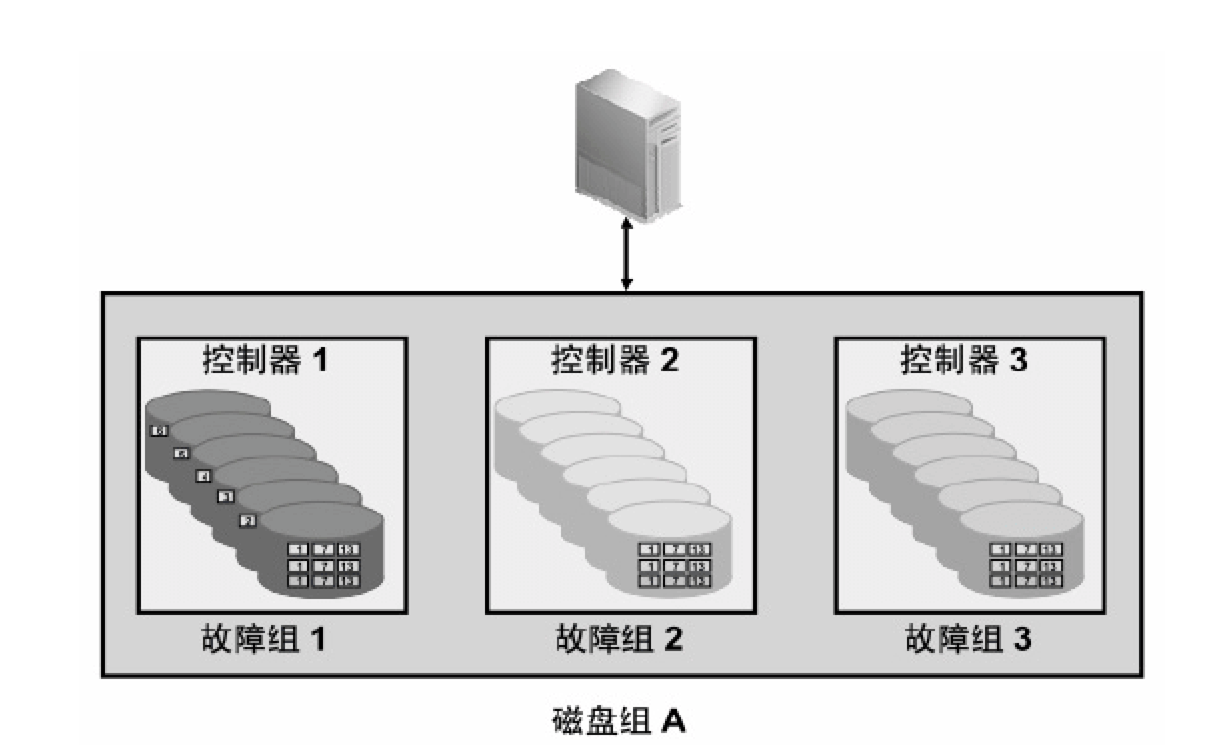


图 4-4　ASM 故障组

故障组是某个特定磁盘组中的一组磁盘，共享一个需要容错的公用资源。例如，故障组可以是连接到公用 SCSI 控制器的一组 SCSI 磁盘。尽管每个单独的磁盘都功能正常，但控制器的故障将导致 SCSI 总线上的所有磁盘都不可用。故障组的构成视站点而定。这主要取决于站点的容错模式。默认情况下，ASM 将每个磁盘分配至各自的故障组。创建磁盘组或将磁盘添加至磁盘组时，管理员可能会指定自己的磁盘分组方法，并将磁盘划分到相应的故障组中。确定了故障组之后，ASM 可以优化文件布局，降低因共享资源故障而导致的数据不可用。

磁盘组镜像，ASM 具有三种磁盘组类型，可支持不同镜像类型：

- 外部冗余：不提供镜像。如果使用硬件镜像，或可以容忍磁盘故障所导致的数据丢失则使用外部冗余磁盘组。故障组不与这些类型的磁盘组一起使用。
- 正常冗余：支持双向镜像。
- 高冗余：提供三向镜像。

ASM 不是镜像磁盘，而是镜像区。因此，只有磁盘组需要备用容量。某一磁盘发生故障时，ASM 将从磁盘组中的其他正常磁盘读取镜像内容，然后自动在正常磁盘中重建故障磁盘的内容。这就将集中到故障磁盘的 I/O 分布到其他几个磁盘。ASM 将文件的一个主分配单元分配至磁盘组中的一个磁盘时，会将该分配单元的一个镜像副本分配至该磁盘组的另一个磁盘。给定磁盘中的主分配单元可以在该磁盘组的一个伙伴磁盘上拥有自己的镜像副本。ASM 将确保主分配单元及其镜像副本始终位于不同的故障组中。如果为磁盘组定义了故障组，则 ASM 可以对同一故障组中多个磁盘发生的同步故障进行容错处理，如图 4-5 所示。

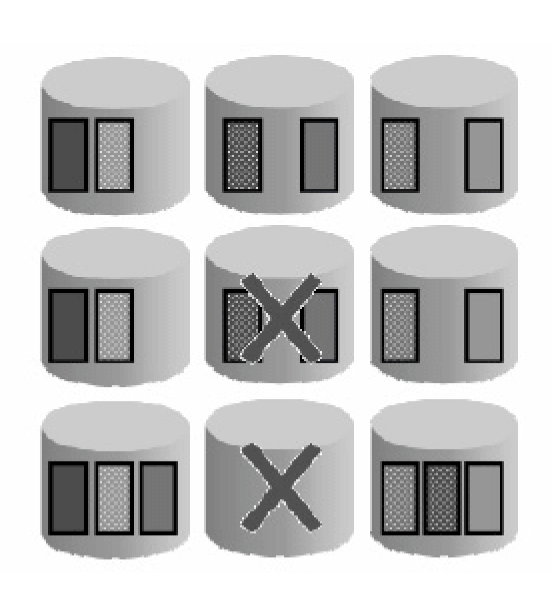


图 4-5　ASM 镜像

数据存储在 ASM 磁盘组中会被 ASM 自动打散分布在每个成员磁盘中，这个过程称为磁盘组的动态重新平衡。使用 ASM，可以非常轻松地完成重新平衡过程，无须 DBA 或系统管理员进行任何干预。无论何时添加或删除磁盘，ASM 均可以自动重新平衡磁盘组。然而，当磁盘组由于错误而被删除时会延迟进行重新平衡。ASM 使用索引技术将区分布在各个可用磁盘上，只需移动与添加或删除的存储量成一定比例的数据量，就可以在磁盘组的各个磁盘上重新均匀分配文件并在这些磁盘之间保持 I/O 负载平衡，而不必重新条带化所有数据。

由于可以保持 I/O 负载平衡，因此无论何时分配文件，也不论存储配置何时发生更改，DBA 都无须搜索磁盘组中的热点，也不用手动移动数据来恢复 I/O 负载平衡。但是，因为数据库需要再同步高速缓存的 ASM 元数据，因此在不繁忙的时段执行重新平衡产生的影响较小。如果同时添加或删除多个磁盘，则使用 ASM 会更高效，因为可通过一个操作实现各磁盘的重新平衡。这可避免数据发生不必要的移动。使用此技术，可以轻松地实现数据的联机移植。只需一个操作便可添加新磁盘，删除旧磁盘。通过设置 ASM_POWER_LIMIT 变量，可以控制重新平衡操作在系统上分配的负载量。其值的范围介于 1 和 11 之间，值越小，负载越小；值越高，负载越大，完成得越快。

4.5 磁盘组的管理

ASM 实例的主要目标是管理磁盘组和保护其中的数据。ASM 实例还可以向数据库实例传递有关文件布局的信息。通过这种方式，数据库实例可以直接访问磁盘组中存储的文件。系统提供了多个磁盘组管理命令。使用这些命令需要具有 SYSASM 或 SYSDBA 权限，并且必须从 ASM 实例发出这些命令。既可以添加新的磁盘组，也可以修改现有的磁盘组，在其中添加新磁盘、删除现有磁盘和执行许多其他操作，还可以删除现有的磁盘组。

【示例 4-1】创建和删除磁盘组

```
drop diskgroup including contents;
create diskgroup dgroupa normal redundancy
failgroup controller1 disk
'/devices/a1' name diska1 size 120g force,
'/devices/a2',
'/devices/a3'
failgroup controller2 disk
'/devices/b1',
'/devices/b2',
'/devices/b3';
```

假定在 ASM 磁盘搜索过程中，在/devices 目录中找到磁盘 A1、A2、A3、B1、B2 和 B3，又假定磁盘 A1、A2 和 A3 位于一个 SCSI 控制器上，而磁盘 B1、B2 和 B3 位于另一个控制器上。幻灯片中的第一个示例 4-1 说明了如何配置名为 dgroupa 的磁盘组，该磁盘组有两个故障组：controller1 和 controller2。该示例中还对磁盘组使用了默认的冗余特性 normal redundancy。

可以根据需要提供磁盘的名称和大小。如果不提供此信息，ASM 将创建一个默认名称，并尝试确定磁盘的大小。如果不能确定大小，则会返回错误。force 表示，即使指定磁盘已作为 ASM 磁盘组的成员进行了格式化，也应该将该磁盘添加到指定磁盘组中。如果对未作为 ASM 磁盘组成员进行格式化的磁盘使用 force 选项，则会返回错误。如第二个语句所示，可以删除磁盘组及其所有文件。如果除内部 ASM 元数据外，磁盘组中还包含任何其他文件，则必须指定 including contents 选项，以避免出现误删错误。必须先装载磁盘组，然后才能将其删除。确保磁盘组中的任何文件均未打开后，将该组及其所有驱动器一起从磁盘组中删除。然后，改写每个磁盘的标头，去除其中的 ASM 格式信息。

【示例 4-2】将磁盘添加至磁盘组

```
alter diskgroup dgroupa add disk
'/dev/rdsk/c0t4d0s2' name a5,
'/dev/rdsk/c0t5d0s2' name a6,
'/dev/rdsk/c0t6d0s2' name a7,
'/dev/rdsk/c0t7d0s2' name a8;

alter diskgroup dgroupa add disk '/devices/a*';
```

如何将磁盘添加到磁盘组。可以执行 alter diskgroup add disk 命令将磁盘添加到磁盘组。第一个语句将四个新磁盘添加到 dgroupa 磁盘组。第二个语句演示搜索字符串的相互影响。请考虑以下配置：

- /devices/a1 是磁盘组 dgroupa 的成员。
- /devices/a2 是磁盘组 dgroupa 的成员。
- /devices/a3 是磁盘组 dgroupa 的成员。
- /devices/a4 是候选磁盘。

第二个命令将 a4 添加到磁盘组 dgroupa 中。由于其他磁盘已经是 dgroupa 磁盘组的成员，因此即使它们与搜索字符串匹配，第二个语句仍会忽略这些磁盘。向磁盘组添加磁盘时，ASM 实例将确保该磁盘是可寻址的并且可用，然后才会对该磁盘进行格式化并使其重新平衡。由于需要将所有文件的区移到新磁盘上，因此重新平衡过程非常耗时。



重新平衡不会妨碍任何数据库操作。重新平衡进程主要会对系统上的 I/O 负载产生影响。重新平衡的强度越高，它加在系统上的 I/O 负载也就越大。这样，可供数据库 I/O 使用的 I/O 带宽就越少。

其他管理命令如下所示。

【示例 4-3】从 dgroupa 中删除一个磁盘

```
alter diskgroup dgroupa drop disk a5;
```

【示例 4-4】通过单个命令添加和删除磁盘

```
alter diskgroup dgroupa
drop disk a6
add failgroup fred
disk '/dev/rdsk/c0t8d0s2' name a9;
```

【示例 4-5】取消磁盘删除操作

```
alter diskgroup dgroupa undrop disks;
```

示例 4-3 显示如何从 dgroupa 磁盘组中删除一个磁盘。示例 4-4 显示如何使用单个命令添加和删除磁盘。本例的最大优点在于在命令完成前不会启动重新平衡。

示例 4-5 显示如何取消磁盘删除操作。undrop 命令只对磁盘的暂挂删除操作有效，对已完成的删除操作没有任何影响。

【示例 4-6】根据需要重新平衡 dgroupb 磁盘组

```
alter diskgroup dgroupb rebalance power 5;
```

此命令通常不是必需的，因为在添加、删除磁盘或调整磁盘大小时会自动执行此操作。不过，如果希望使用 power 子句来覆盖初始化参数 ASM_POWER_LIMIT 定义的默认速度，则该命令非常有用。通过在命令中重新输入新的级别，可以更改正在进行的重新平衡操作的强度。如果强度级别为 0，则重新平衡操作将中断，直到重新隐式或显式调用该命令。

【示例 4-7】卸载 dgroupa

```
alter diskgroup dgroupa dismount;
```

使用 mount 和 dismount 选项，可以指定可供数据库实例使用或不可供其使用的一个或多个磁盘组。当实例故障转移到其他节点时这种在支持单个实例的集群 ASM 环境中手动执行卸载和装载的功能很有用。

4.6 ASM 磁盘组兼容性

适用于 ASM 磁盘组的兼容性有两种：一种涉及说明磁盘组的持久数据结构，一种涉及客户机（磁盘组的使用者）的功能。这两个属性分别被称作 ASM 兼容性和 RDBMS 兼容性。可以单独控制每个磁盘组的兼容性。这是支持包含 Oracle Database 10g 和 Oracle Database 11g 的磁盘组的异构环境所必需的。这两种兼容性设置是每个 ASM 磁盘组的属性。

- RDBMS 兼容性是指允许该实例装载磁盘组的 RDBMS 实例的最低兼容版本。该兼容性确定了 ASM 实例与数据库（RDBMS）实例间交换消息的格式。ASM 实例可以支持以不同兼容性设置运行的不同 RDBMS 客户机。每个实例的数据库兼容版本设置必须高于或等于该数据库使用的所有磁盘组的 RDBMS 兼容性。数据库实例与 ASM

实例通常在不同的 Oracle 主目录中运行。这表示数据库实例与 ASM 实例所运行的软件版本可以不同。数据库实例第一次连接到 ASM 实例时，系统会协定这两个实例都支持的最高版本。

- 数据库的兼容性参数设置、数据库的软件版本以及磁盘组的 RDBMS 兼容性设置确定了数据库实例能否装载指定的磁盘组。
- ASM 兼容性是指控制磁盘上 ASM 元数据的数据结构格式的持久兼容性设置。磁盘组的 ASM 兼容性级别必须始终高于或等于同一磁盘组的 RDBMS 兼容性级别。ASM 兼容性只与 ASM 元数据的格式相关。文件内容的格式取决于数据库实例。例如，可以将某个磁盘组的 ASM 兼容性设置为 11.0，而将该磁盘组的 RDBMS 兼容性设置为 10.1，这表示该磁盘组只能由软件版本为 11.0 或更高的 ASM 软件管理，而软件版本高于或等于 10.1 的任何数据库客户机都可以使用该磁盘组。仅当持久磁盘结构或消息传送协议发生更改时，才需要提高磁盘组的兼容性。但是，提高磁盘组兼容性是一个不可逆的操作。可以使用 create disk group 命令或 alter disk group 命令来设置磁盘组兼容性。



除磁盘组兼容性之外，兼容参数（数据库兼容版本）决定了所支持的功能。兼容参数适用于数据库或 ASM 实例，具体取决于 instance_type 参数。例如，将该参数设置为 10.1，将禁止使用 Oracle Database 11g 中引入的任何新功能（磁盘联机/脱机、可变区等）。

4.7 ASMCMD 程序

关于 ASM 的日常管理，相信使用率最高的工具非 ASMCMD 莫属。ASMCMD 是一个命令行实用程序，可用于查看和操纵 ASM 磁盘组中的文件和目录。该实用程序可以列出磁盘组内容、执行搜索、创建和删除目录和别名以及显示空间使用情况等。ASMCMD 可以处理 ASM 文件、目录和别名。在 ASM 中创建的所有文件都拥有一个系统生成的文件名，这也被称为全限定的文件名。在本地文件系统中它与完整的路径名相同。与其他文件系统中的目录相同，ASM 目录是一种文件容器，可以位于其他目录的树结构中。全限定的文件名可以表示目录的层次，其中加号（+）表示根目录。

【示例 4-8】全限定的文件名

```
asmcmd> ls -l +dgroup1/orcl/datafile
type      redund  striped  time            sys  name
datafile  mirror  coarse   oct 05 21:00:00  y   hrapps.257.570923611
```

可以使用 ASMCMD mkdir 命令创建自己的目录，作为系统生成的目录的子目录：

【示例 4-9】使用 mkdir 命令创建自己的目录

```
asmcmd> mkdir +dgroup1/sample/mydir
```

除了基本的查看磁盘组状态的命令之外，ASMCMD 还有如下几项强大的功能。

- ASMCMD 可以执行 ASM 元数据备份和还原功能。这样一来，就可以使用完全相同的模板和别名目录结构重新创建先前存在的 ASM 磁盘组。
- lsdsk 命令可列出 ASM 磁盘信息。此命令可在两种模式下运行：连接和非连接。在连接模式下，ASMCMD 使用 V$和 GV$视图来检索磁盘信息。在非连接模式下，ASMCMD 使用 ASM 磁盘字符串来限制搜索集，对磁盘头进行扫描以检索磁盘信息。连接模式始终为首选模式。
- 修复损坏的块是一种自动在正常冗余或高冗余磁盘组上运行的新功能。当正常读取 ASM 磁盘组失败并出现 I/O 错误时，ASM 尝试通过读取镜像副本，然后再将内容写入到该块中来修复，如果无法有效读取副本，则会进行重新定位。仅对所读取的块自动执行整个过程。ASM 磁盘组中的一些块和区可能很少被读取。一个主要示例是二级区。ASMCMD 的修复命令旨在触发对这些区的读取操作，因此导致的 I/O 故障可以启动自动块修复过程。如果存储数组返回物理块错误，用户可以使用 ASMCMD 修复界面，ASMCMD 修复随后会启动对该块的读取操作，从而触发修复过程。

4.8 小结

通过本章内容读者可以了解 ASM 组件的体系结构和特点。它是为了简化 Oracle 数据库的管理而推出来的一项新功能，是 Oracle 自己提供的卷管理器，主要用于替代操作系统所提供的 LVM。它不仅支持单实例，同时对 RAC 的支持也非常好。ASM 可以自动管理磁盘组并提供有效的数据冗余功能。使用 ASM（自动存储管理）后，数据库管理员不再需要对 Oracle 中成千上万的数据文件进行管理和分类，从而简化了 DBA 的工作量，可以使得工作效率大大提高。

作为数据库管理员，除了日常管理好数据库之外，还有一项重要的工作需要完成，就是备份恢复，以此应对数据库故障或者灾难的发生。从重要性的角度来讲，数据库备份重于一切，是数据库的生命线。关于数据库备份恢复的相关知识，将在下一章节向读者阐述。

第 5 章

通过配置实现数据库可恢复

如果数据库出现问题，需要能够对其进行恢复。如果创建数据库备份，则可防止出现诸如介质故障、用户错误和应用程序错误之类的问题。介质错误会因硬件级故障而引发数据问题；控制器故障或磁盘驱动器故障会导致不太明显的错误或明显的错误。如果用户发出不该发出的命令，则还会导致数据错误。应用程序故障也会导致这些相同类型的错误。

备份还可用来保留数据。可能需要在一个有意义的确定时间点创建一个数据库副本，并将其存储很长一段时间。这可能用于将来的还原，也可能只为符合规定。还可以使用备份和恢复工具将数据移到其他数据库，甚至移到其他位置。对于备份数据库，要在其他位置将其还原，使用数据库备份是一种有效的方法。

5.1 备份恢复任务

如果数据库丢失会造成非常严重的后果，那么制订一个强大的备份和恢复计划非常重要。该计划包含下列任务：

- 配置：需要针对的环境配置备份和恢复环境。这包括配置一些设置，如备份的目标、删除前备份保留的时间以及加密。
- 计划：可以调度备份，这样便无须手动启动这些备份。这很有用，因为创建备份的最佳时间通常是在空闲时段内。
- 测试：应对发生数据损坏或丢失的情形进行计划和测试，并使用创建的备份进行恢复。应定期执行此操作，这样便可确定是否成功备份了所需的内容。
- 监视：执行备份需要一些资源，这可能会影响数据库中的其他操作。应监视备份和恢复任务，并确保这些任务正在有效地运行。
- 还原：需要依靠备份时，必须从中还原数据。此时会将文件存入数据库，并且通常会将数据库状态置为过去的一个时间点。
- 恢复：从备份还原文件后，如果需要将数据库恢复到离当前较近的时间点（或恢复到当前时间点），则需要执行恢复。这就是将重做数据应用于还原数据的过程。

5.2 Oracle 的备份恢复方案

下面是常用的备份和恢复解决方案。

- Recovery Manager：执行备份和恢复的命令行工具。使用 RMAN 时，可使用的一些主要功能有：
  - 增量备份：一种只将自上次增量备份后发生更改的数据块写入备份的备份类型。
  - 块介质恢复：一种恢复特定数据块的方法，与恢复整个表（使用数据泵）或数据文件（使用 RMAN）相对。
  - 未使用的块压缩：一种不将从未使用过的块写入备份的节省空间的方法。
  - 二进制压缩：一种使用已知算法压缩备份文件的节省空间的功能（与 Linux 中的 zip 等实用程序类似）。
- 备份加密：保护所创建的备份的安全设备。
- 数据泵：在 OS 文件中导出和导入表数据的命令行工具。

5.3 配置数据库使其可恢复

在 ARCHIVELOG 模式下运行数据库时，数据丢失后可使用更多的恢复选项，其中包括数据库或某些表空间的时间点恢复。建议利用快速恢复区存储尽可能多的与备份和恢复相关的文件，其中包括磁盘备份和归档重做日志。有些 Oracle DB 备份和恢复功能（如 Oracle 闪回数据库和可靠还原点）要求使用快速恢复区。

当对数据库中的数据进行修改后，重做数据会写出到联机重做日志文件中。指定系统在给定时间向其写入数据的文件。当此文件写满后，归档进程（ARCn）会将该联机日志文件复制到其他位置，作为该文件的归档，保留时间由系统管理员自行决定。一般来讲，系统管理员会根据企业的整体要求，将归档文件转移至备份数据库中保留五年或更长时间，本地的归档文件可以在将其备份到数据库之后进行删除。这便提供了更多的恢复机会，因为可以保存、备份和还原生成的所有归档重做日志。因为系统以循环方式重用联机重做日志文件，所以有一个协议用于控制何时可以重用文件。在 ARCHIVELOG 模式下，数据库只在联机重做日志文件归档后向其写入数据。这样可确保每个重做日志文件都有机会得以归档。

将数据库置于 ARCHIVELOG 模式可防止重做日志在归档之前被覆盖。

要发出 SQL 命令将数据库置于 ARCHIVELOG 模式，数据库必须处于 MOUNT 模式。为了变成 MOUNT 状态，数据库必须处于 SHUTDOWN 状态；如果数据库当前处于打开状态，必须将其关闭，然后进行装载。下面显示的命令可以关闭打开的数据库，将其置于 ARCHIVELOG 模式，然后将其打开。

【示例 5-1】利用 SQL 命令将数据库置于 ARCHIVELOG 模式

```
sql> shutdown immediate
sql> startup mount
sql> alter database archivelog;
sql> alter database open;
```

数据库处于 NOARCHIVELOG 模式（默认模式）时，只能恢复到最后一次备份时的状态。在该备份之后执行的所有事务处理都会丢失。

在 ARCHIVELOG 模式下，可一直恢复到最后一次提交时的状态。多数生产数据库都在 ARCHIVELOG 模式下运行。



请在切换到 ARCHIVELOG 模式后备份数据库，原因是数据库只能从在该模式下执行的最后一次备份进行恢复。

在指定归档日志文件所写入到的位置时，有两种模型可供选择，如图 5-1 所示。

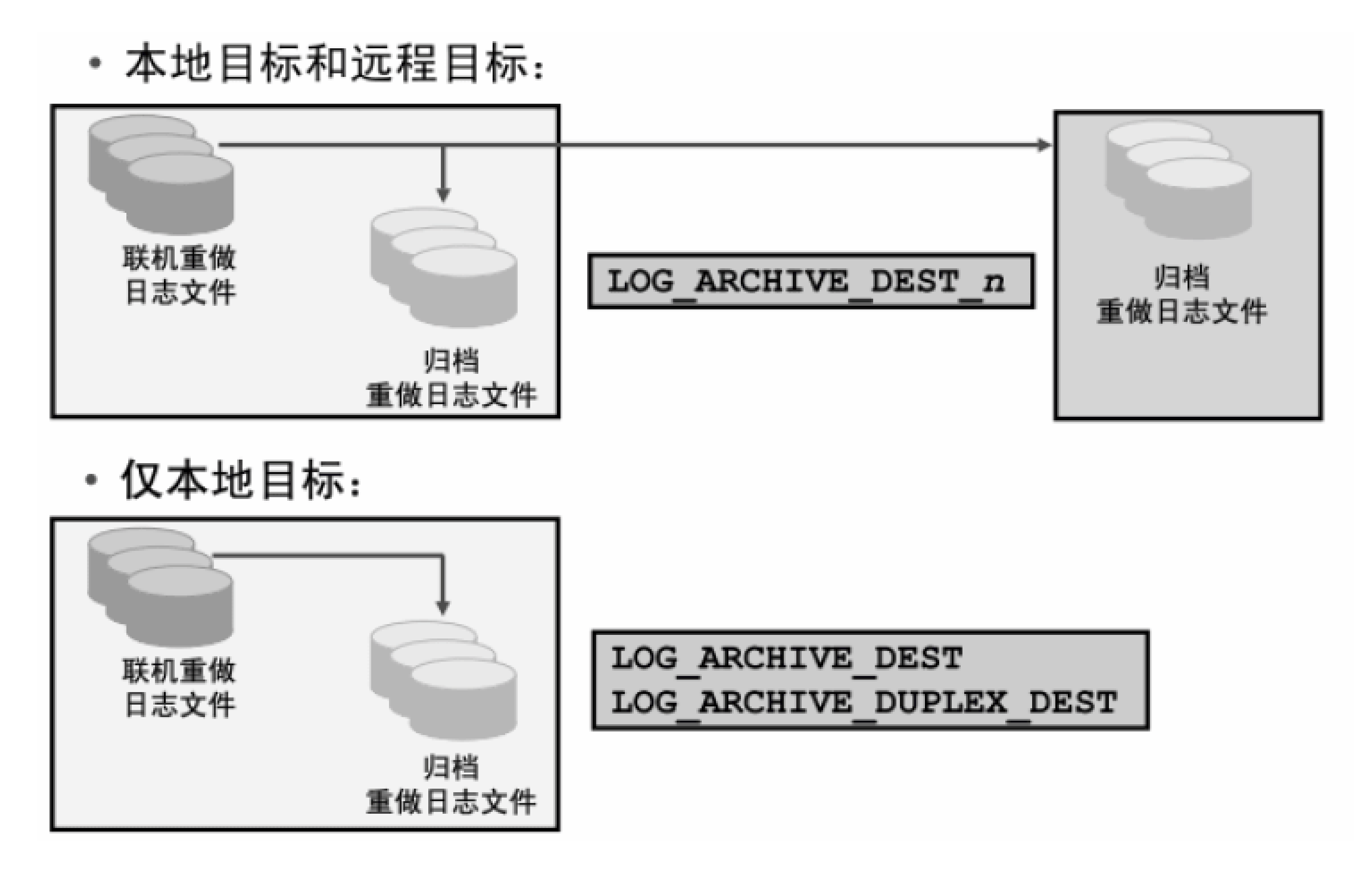


图 5-1　归档文件写入位置

（1）本地目标和远程目标：通过设置 LOG_ARCHIVE_DEST_n 初始化参数的集合指定本地目标和远程目标。有十个参数，因此 n 可以为 1 到 10。为了指定本地存储位置，应为其中一个变量的值提供本地目录名称，即提供“LOCATION=”字符串。如示例 5-2，要指定 /disk3/arch 目录，可按如下方式设置其中一个变量。

【示例 5-2】指定本地存储位置

```
log_archive_dest_1 = 'location=/disk3/arch'
```

如果要为备用数据库指定远程位置，请在值中使用 SERVICE 关键字，如示例 5-3 所示，其中 standby1 是备用数据库实例的服务名称。

【示例 5-3】指定远程位置

```
log_archive_dest_2 = 'service=standby1'
```

（2）仅本地目标：用于指定目标的另一选项，仅支持本地磁盘位置。将 LOG_ARCHIVE_DEST 和 LOG_ARCHIVE_DUPLEX_DEST 参数设置为本地磁盘目录。因此，最多可以有两个归档日志文件位置，如示例 5-4 所示。

【示例 5-4】指定仅本地磁盘位置

```
log archive dest = '/disk1/arch'
log_archive_duplex_dest = '/disk2/arch'
```

Oracle 建议使用 LOG_ARCHIVE_DEST_n 方法，因为此方法在目标类型以及目标数量方面具有很大的灵活度。

如果为归档日志文件指定了多个目标，则应指定至少有多少个目标获得成功才认为归档是成功的。请使用 LOG_ARCHIVE_MIN_SUCCEED_DEST 初始化参数执行此操作。将其设置为必须成功接收归档日志文件的目标数量。只有满足此数量，才可以重用联机日志文件。如图 5-2 所示，联机日志高可用有三个指定的目标：两个是本地目标，一个是远程目标。

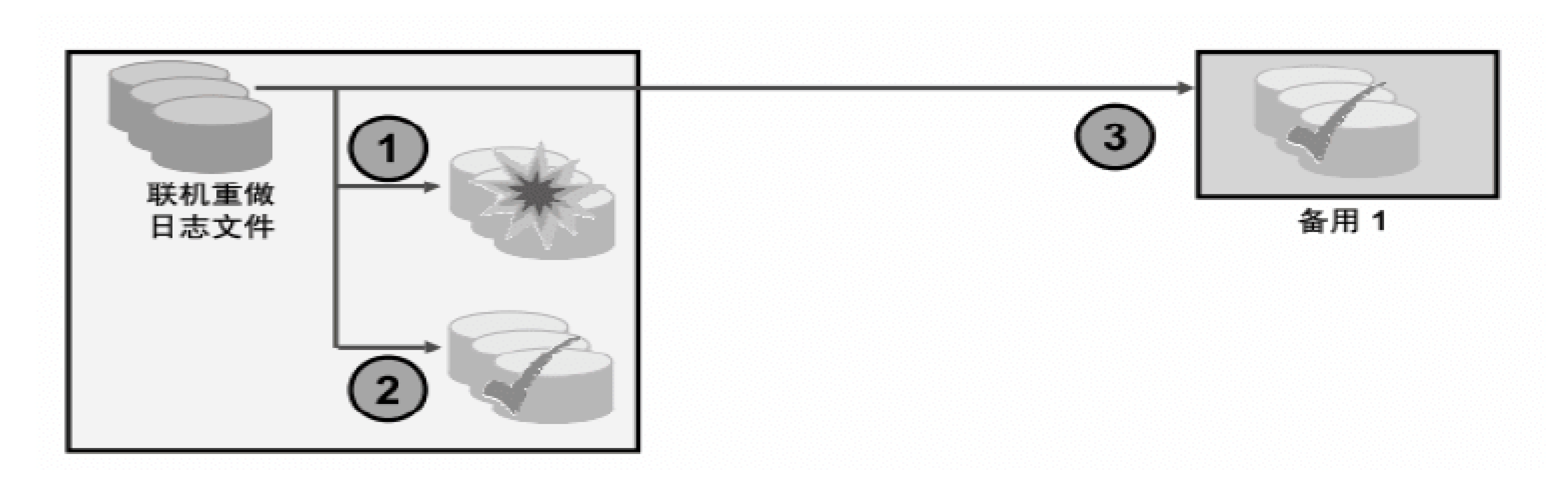


图 5-2　联机日志高可用

LOG_ARCHIVE_MIN_SUCCEED_DEST 设置为 2，表示只要至少有两个目标成功，即可覆盖联机重做日志文件。图 5-2 中显示目标 1 已失败。这并不能使数据库停止，因为其中两个已成功。可以在上一幻灯片中所述的任一模型中使用此参数。如果在 LOG_ARCHIVE_DEST_n 模型中使用此参数，则此参数的值可以介于 1 到 10 之间。如果在 LOG_ARCHIVE_DEST 模型中使用此参数，则值可以为 1 或 2，因为在这种情况下只能指定两个目标。

指定 MANDATORY 和 OPTIONAL 定义目标时，可以指定该目标为必需目标。在指定位置后，通过指定 MANDATORY 或 OPTIONAL 关键字可实现上述目的，如示例 5-5 所示。

【示例 5-5】指定必需目标

```
log_archive_dest_1 = 'location=/disk3/arch mandatory'
```

默认值为 OPTIONAL。系统尤为关注必需目标。如果任一必需目标失败，则 Oracle DB 认为日志归档尚未成功，不允许覆盖联机重做日志文件。在此情况下，它将忽略 LOG_ARCHIVE_MIN_SUCCEED_DEST 参数。LOG_ARCHIVE_DEST 指定的任何目标都是必需的。如果 LOG_ARCHIVE_MIN_SUCCEED_DEST= 1，则 LOG_ARCHIVE_DUPLEX_DEST 声明的所有目标都是可选的，如果 LOG_ARCHIVE_MIN_SUCCEED_DEST= 2，则目标都是必需的。

5.4 使用快速恢复区

快速恢复区是 Oracle DB 中所有与恢复相关的文件和活动的统一存储位置。从介质故障中完全恢复数据库所需的所有文件都包含在快速恢复区中。与恢复相关的文件有两种类型：永久文件和临时文件。永久文件由实例主动使用，临时文件只有在执行某类恢复操作时才需要，下面是几个所谓的永久文件。

（1） 控制文件：根据几个初始化参数的设置，控制文件的副本是在创建新数据库或控制文件时在快速恢复区中创建的。有关详细信息，请参阅《Oracle Database SQL Language Reference》中 CREATE CONTROLFILE 命令的“Semantics”一节。

（2）联机重做日志文件的多路复用副本：可包含每个重做日志组的镜像副本。创建数据库时，可以使用 LOGFILE 子句指定联机重做日志文件的位置。如果不包含该子句，系统会根据下列初始化参数的值设置位置：

- DB_CREATE_ONLINE_LOG_DEST_n：如果设置了这些变量中的一个或多个，则只使用这些位置。
- DB_CREATE_FILE_DEST：如果设置了此参数，则该位置是主要文件位置。
- DB_RECOVERY_FILE_DEST：如果除 DB_CREATE_FILE_DEST 外还设置了此参数，则此位置将用作镜像。

有关这些变量如何影响联机重做日志的位置的详细信息，请参阅《Oracle Database SQLLanguage Reference》中 CREATE DATABASE 语句的 LOGFILE 子句。

在闪回恢复区内会存放几类临时项目：

- 归档重做日志文件：配置了快速恢复区后，LOG_ARCHIVE_DEST_10 会被自动设置为快速恢复区位置。归档后台进程会在快速恢复区和配置的其他 LOG_ARCHIVE_DEST_n 位置中创建归档重做日志文件。如果未定义 LOG_ARCHIVE_DEST_n 位置，则归档重做日志文件的默认位置是快速恢复区。
- 闪回日志：启用闪回数据库时会生成闪回日志。
- 控制文件自动备份：由 RMAN 创建的控制文件自动备份和由 OracleDB 服务器生成的自动备份的默认位置是快速恢复区。
- 数据文件副本：BACKUPASCOPY 命令在快速恢复区中创建映像数据文件副本。
- RMAN 文件：快速恢复区是 RMAN 用于备份和通过恢复操作从磁带还原归档日志内容的默认位置。

关于快速恢复区，有两个重要的参数需要引起读者注意，可以使用以下必需参数定义快速恢复区：

- DB_RECOVERY_FILE_DEST_SIZE：必须定义磁盘限制，这是允许快速恢复区使用

的空间量。通过设置限制可留出一部分磁盘空间作为它用，而不被快速恢复区所专用。磁盘限制大小的基本建议值等于数据库大小、增量备份大小和尚未复制到磁带的所有归档日志文件的大小的总和。快速恢复区的最小大小应至少足够放得下尚未复制到磁带的归档重日志文件。快速恢复区的大小与实施的备份策略和其他选项有关。闪回数据库设置点、更改块跟踪文件以及多级增量备份都会对快速恢复区的大小产生影响。

- DB_RECOVERY_FILE_DEST：快速恢复区说明中包含一个位置，这个位置是创建文件的有效目标。RMAN 每次在快速恢复区中创建文件时，都会更新磁盘上不再需要的文件的列表。根据 DB_REOVERY_FILE_DEST_SIZE 的值，当快速恢复区因没有可删除的文件而出现空间紧缺或空闲空间不足的情况时，系统会警告存在磁盘空间不足的危险。这时，OracleDB 服务器和 RMAN 会继续在快速恢复区中创建文件，直到达到 100%磁盘限制。设置 DB_RECOVERY_FILE_DEST_SIZE 时，必须分配足够的空间来存放恢复文件，包括等待备份到磁带的备份。为了提供空闲空间，过时或已备份到磁带的文件都可能被删除。将某一文件写入到快速恢复区时，如果需要为该文件腾出空间，OracleDB 服务器会删除过时文件列表中的某个文件。在快速恢复区中写入和删除文件时，会向预警日志写入通知。



当快速恢复区的已用空间达到 85% 时，会发出警告；当已用空间达到 97% 时，会发出严重警告。这些都是内部设置，无法更改。

【示例 5-6】预警日志输出示例

```
warning: db recovery file dest size of 52428800 bytes is
100.00% used, and has 0 remaining bytes available.
```

【示例 5-7】发出以下查询来确定要执行的操作

```
sql> select object type, message type, message level,
reason ,suggested_action
from dba_outstanding_alerts;
```

可以选择增加额外的磁盘空间、将文件备份到三级存储设备、使用 RMAN 从快速恢复区删除文件或考虑更改 RMAN 保留策略。要避免快速恢复区出现空间不足的情况，请按需要或视情况执行下列步骤：

- 使用 RMAN 从快速恢复区删除不需要的文件。
- 使用 RMAN 经常备份快速恢复区。
- 更改 RMAN 保留策略，缩短备份的保留期。
- 更改 RMAN 归档日志删除策略。
- 如果经常出现空间不足的情况，请增加磁盘空间并加大。

DB_RECOVERY_FILE_DEST_SIZE 数据库初始化参数值。OracleEnterpriseManager 不会报告快速恢复区在磁盘上使用的空间量或者快速恢复区目录树已用的空间量，但会报告

RMAN 认为位于目录中的文件的大小。所以不要将任何文件放在此不受 RMAN 管理的区域。如果使用 RMAN 之外的工具从此区域删除任何文件，请使用 RMAN 从目录中删除文件项。例如，要备份快速恢复区中的归档日志文件，然后在成功备份这些文件后将其删除，应使用 RMAN 命令。

【示例 5-8】删除文件

```
backup archivelog all delete all input;
```

即使使用 RMAN 之外的备份解决方案，仍需要使用 RMAN 从快速恢复区删除这些文件。备份归档日志文件并将这些文件从磁盘中删除后，应使用 RMAN 的 CROSSCHECK 和 DELETE 命令从快速恢复区回收归档日志空间。这个操作应定期执行或在每次备份后执行。还可以使用 OracleEnterpriseManager 的“ManageBackups（管理备份）”页来管理备份。可以在该页上执行交叉检查操作，还可以删除过期和过时的备份。

以上关于快速恢复区的内容了解过之后，总结下来将快速恢复区用于与恢复相关的文件具有以下优点：

- 便于确定数据库备份的位置。
- 自动管理为恢复文件分配的磁盘空间。

5.5 配置备份规范

一般来说 Oracle 数据库的日常备份工作应由 RMAN 工具来完成，本节将主要针对 RMAN 讲解备份规范化设计。

备份是数据库中数据的副本，可用来重建数据。使用 RMAN 创建的备份可以是映像副本，也可以是备份集。使用 RMAN 执行备份时，可指定：

- 要执行的备份类型：可以对整个数据库执行备份，从而包括文件中所有已使用的数据库块（FULL 备份）或增量备份（INCREMENTAL）。如果已启用 CONFIGURECONTROLFILEAUTOBACKUP，则 RMAN 会在执行 BACKUP 命令后，自动备份控制文件和当前服务器参数文件。
- 要备份的内容：数据库备份的有效值包括 DATABASE、DATAFILE、TABLESPACE、ARCHIVELOG、CURRENTCONTROLFILE 以及 SPFILE。RMAN 还具有附加命令，可使用这些命令将备份文件移到磁带。
- 是创建映像副本（ASCOPY）还是创建备份集（ASBACKUPSET）。
- 备份片段的文件名格式和位置（FORMAT）。
- 应从备份集中排除哪些数据文件或归档重做日志（SKIP）。
- 成功创建备份集后应删除的输入文件（DELETEINPUT）。
- 指定介质管理库（MML）如何执行文件复制的代理选项。BACKUP 命令的 PROXY

选项提供了一种方法，可以用来减轻 RMAN 工作，而无须了解 MML 控制的介质是如何工作的。

5.5.1 备份目标

备份可以写入到指定的磁盘目录、介质管理库或快速恢复区。指定磁盘目录或快速恢复区意味着备份会移到硬盘介质。通常，为了保持磁盘空间的可用性，定期通过介质管理接口将备份脱机移到磁带中。只要磁盘目录已存在，任何磁盘目录都可以指定为备份目标。如果配置了快速恢复区，很多备份和恢复任务都变得非常简单。Oracle DB 服务器会自动指定文件，在空间紧缺时会删除过时的文件。

5.5.2 配置 RMAN 的永久性设置

为了简化使用 RMAN 进行备份和恢复的过程，可以使用 RMAN 为每个目标数据库设置一些永久性配置设置。这些设置可控制 RMAN 行为的多个方面。可以保存永久性配置信息，如通道参数、并行度和 RMAN 资料档案库中的默认设备类型。这些配置设置始终存储在控制文件和恢复目录数据库中（如果存在）。这些设置都提供了默认值，可以立即使用 RMAN。但是，当制定更高级的备份和恢复策略时，可能要更改这些设置才能实施该策略。使用 CONFIGURE 命令可为 RMAN 备份、还原、复制和维护作业配置永久性设置。这些设置在所有 RMAN 会话中均有效，直到清除或更改了配置为止。



可以使用 SET 命令在 RMAN 作业（或会话）的持续期间内更改该作业（或会话）的这些配置设置。

5.5.3 控制文件自动备份

要轻松地从所有控制文件副本丢失中恢复过来，应将 RMAN 配置为自动备份控制文件。控制文件的自动备份与备份命令明确请求的对当前控制文件的备份无关。如果在 NOCATALOG 模式下运行 RMAN，则强烈建议激活控制文件自动备份功能。如果丢失了控制文件，就可能无法恢复数据库。

要配置控制文件自动备份，请使用 Oracle Enterprise Manager 或使用以下 RMAN 命令修改数据库的备份策略：

```
CONFIGURE CONTROLFILE AUTOBACKUP ON
```

默认情况下，控制文件自动备份处于禁用状态。如果启用了控制文件自动备份，则 RMAN 在出现以下情况时就会自动备份控制文件和当前服务器参数文件（如果该文件用来启动数据库）：

- 在 RMAN 资料档案库中记录了成功的备份。
- 数据库的结构更改影响了控制文件的内容，因此必须进行备份。

- 对于所有设备类型，控制文件自动备份文件名都有一个默认格式%F，因此 RMAN 不需要资料档案库就可以推断出文件位置并还原该文件。该变量格式会转换为 c-IIIIIIIIII-YYYYMMDD-QQ。其中，IIIIIIIIII 代表 DBID，YYYYMMDD 是生成备份那天的时间戳，QQ 是十六进制序列，从 00 开始，最大值为 FF。

5.5.4　配置设备和分配通道

【示例 5-9】配置备份设备可以使用

```
configure controlfile autobackup format for device type
```

type TO 'string' 命令更改默认格式。字符串的值必须包含替代变量 %F，不能包含其他替代变量。

【示例 5-10】更改默认格式

```
configure controlfile autobackup format  for device type disk to
'/u01/oradata/cf_orcl_auto_%f';
```

除非另外指明，否则控制文件自动备份存储在快速恢复区中。使用控制文件自动备份，即使无法访问当前控制文件、恢复目录和服务器参数文件，RMAN 仍可恢复数据库。因为用于存储自动备份的文件遵从已知格式，所以 RMAN 可从自动备份中搜索并还原服务器参数文件或控制文件。

使用 RMAN 的 SHOW 命令可查看 RMAN 配置设置。如果连接到目标数据库后执行 SHOW ALL，则只显示特定节点的配置和数据库配置。通过执行带有 CLEAR 选项的 CONFIGURE 命令，可以使用任何 CONFIGURE 命令恢复默认值。

【示例 5-11】使用 SHOW 命令列出当前设置

```
rman> show controlfile autobackup format;
rman> show exclude;
rman> show all;
```

【示例 5-12】使用 CONFIGURE 命令的 CLEAR 选项将任何永久性设置重新设置为默认值

```
rman> configure backup optimization clear;
rman> configure maxsetsize clear;
rman> configure default device type clear;
```

可用使用 CONFIGURE DEVICE TYPE 命令将设备配置为由 RMAN 使用。并行度是可用于对设备进行读取和写入的数据流的数量。当设备由 RMAN 使用时，这可以有效地分配通道的数量。例如，如果介质管理器有两个磁带机可用，则并行度为 2 时使用该介质管理器 BACKUP 命令可以同时使用两个磁带驱动器。要将某个备份分布在多个磁盘中时，磁盘设备类型的并行度也很有用。可以使用如示例 5-13 所示的 PARALLELISM 子句指定设备上使用的并行度。

【示例 5-13】指定设备上使用的并行度

```
configure device type <device> parallelism <n>
```

其中，<n>是并行度值。

备份的输出既可以是备份集，也可以是映像副本。使用 BACKUP TYPE TO 子句配置设备类型的默认值。对于备份集，请指定 BACKUPSET，对于映像副本，请指定 COPY。

【示例 5-14】配置设备的类型

```
rman> configure device type sbt parallelism 3;
rman> configure device type disk
rman> backup type to compressed backupset;
rman> configure device type disk backup type to copy;
```

在BACKUP TYPE TO子句后指定COMPRESSED关键字可以指定对此设备上的备份进行压缩。压缩之后备份文件会变得较小。



只能对备份集应用压缩。

【示例 5-15】配置和分配用于备份的通道

```
rman> configure device type sbt parallelism 1;
rman> configure default device type to sbt;
rman> configure channel device type sbt ...
rman> backup database;
```

【示例 5-16】在 RUN 块内使用 ALLOCATE CHANNEL 命令手动分配通道

```
rman> run
{
allocate channel ch1 device type disk;
backup database plus archivelog;
}
```

5.5.5 配置备份优化

如果启用了备份优化，当完全一样的文件已备份到指定的设备类型时，BACKUP 命令就不会备份这些文件了。如果 RMAN 确定某个文件是完全一样的且已备份，则该文件就是可以跳过的候选文件。但是，RMAN 会通过执行进一步的检查来确定是否跳过该文件，因为 RMAN 用于确定指定设备类型上是否存在充足备份的算法会考虑保留策略和双重备份功能这两个因素。有关 RMAN 用于确定文件是否完全一样的标准以及备份优化算法的详细信息，请参阅《Oracle Database Backup and Recovery User’s Guide》。

在 Oracle Enterprise Manager 中的“Backup Settings（备份设置）”页中，或者通过 CONFIGURE BACKUP OPTIMIZATION ON 命令启用备份优化。默认情况下，备份优化是禁用的。对于 BACKUP RECOVERY AREA|DB_RECOVERY_FILE_DEST 和 BACKUP RECOVERY FILES 命令系统会自动启用备份优化。要覆盖备份优化并备份所有文件（无论是否已更改），请在 BACKUP 命令中指定 FORCE 选项，如示例 5-17 所示。

【示例 5-17】指定 FORCE 选项

```
backup device type sbt backupset all force;
```



FORCE 选项并不适用于恢复区中的文件。

【示例 5-18】使用 Oracle Enterprise Manager 或下列命令永久禁用备份优化

```
configure backup optimization off;
```

5.6 小结

读者可以从本章节的介绍中了解数据库备份的一般配置策略，这部分工作一般是实际生产环境中、数据库上线之初需要布置妥当的。因为一个应用系统数据库如果没有正确合理的数据库备份策略，一旦发生故障或者灾难，宝贵的业务数据将受到损坏并且无法恢复，较好的情况是会仅仅丢失一小部分数据，这对于一个企业或者数据拥有者来说是不可接受的情况，所以备份的重要性不言而喻。那么具体到实际的工作中，备份和恢复如何操作呢？在下一章节，将详细介绍备份和恢复的具体方法，并通过举例让读者了解整个备份恢复的流程和操作方法。

第 6 章

备份与恢复

本章向读者介绍 Oracle 数据库的备份与恢复方法，从具体操作的角度，让读者了解在实际工作中如何处理备份和恢复的工作。这部分的内容也是学习 Oracle 数据库的重点，并且内容也相对较多。作为数据库管理员，备份与恢复的技术能力是必须具备的。

6.1 使用 RMAN 创建备份

RMAN 可以按 RMAN 专用格式存储其备份，这种格式称为备份集。备份集是由一些文件组成的、被称为备份片断的集合，每个备份片断可能包含一个或多个数据库文件备份，如图 6-1 所示。



使用此命令创建备份片断时，FORMAT 参数指定用于创建备份片断文件名的模式。此外，还可通过 ALLOCATE CHANNEL 和 CONFIGURE 命令提供 FORMAT 规范。

【示例 6-1】RMAN 命令

```
rman> backup as backupset
2> format '/backup/df %d %s %p.bus'
3> tablespace hr_data;
```

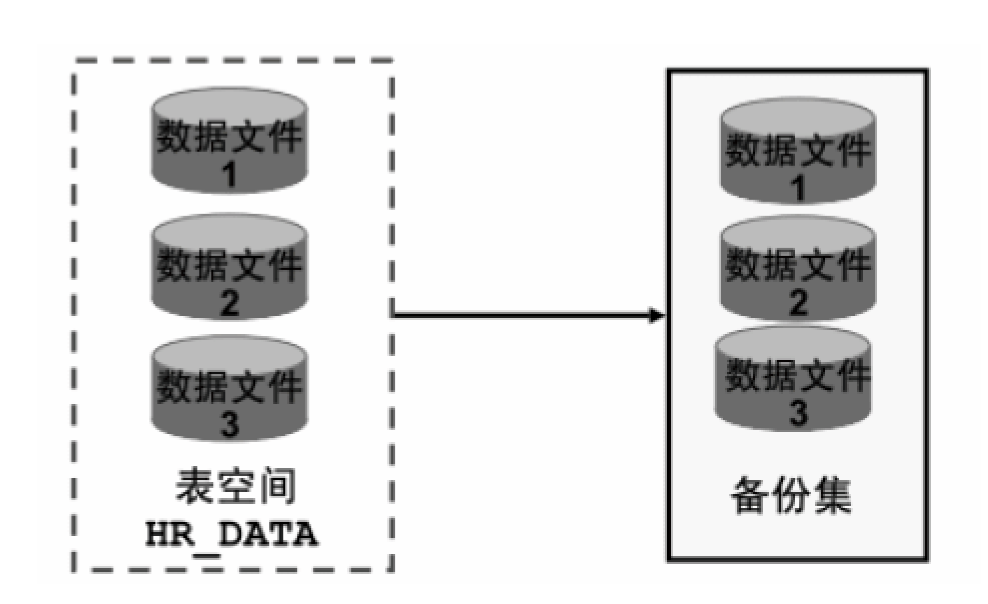


图 6-1 备份集与数据库文件关系

6.1.1　创建映像副本

映像副本是单个数据文件、归档重做日志或控制文件的备份。使用 BACKUP AS COPY 命令或操作系统命令可以创建映像副本。使用 RMAN 的 BACKUP AS COPY 命令创建映像副本时，服务器会话会验证文件中的块，然后在控制文件中记录副本信息。

映像副本具有以下特征：

（1）映像副本只能写入磁盘。处理大型文件时，复制的时间可能很长，但是由于磁盘上提供了副本，因此可明显地减少还原时间。

（2）如果文件存储在磁盘上，在 RMAN 中通过 SWITCH 命令可立即使用文件，这个命令的作用等效于 ALTER DATA BASE RENAME FILE SQL 语句。

（3）在映像副本中，无论块中是否包含数据都会复制所有的块，这是因为 Oracle 数据库进程会复制文件，还会执行其他操作，如检查损坏的块以及在控制文件中注册副本。要加速复制过程，可使用 NOCHECKSUM 参数。默认情况下，RMAN 将计算每个备份块的校验和，并将其与备份一起存储起来。还原备份时，将对校验和进行验证。

（4）映像副本是完全备份的一部分，或是 0 级增量备份的一部分，因为文件副本中始终包含所有块。如果副本与增量备份集一起使用，则必须使用 0 级选项，如图 6-2 所示。

【示例 6-2】创建映像副本

```
rman> backup as copy datafile '/oradata/users_01_db01.dbf';
rman> backup as copy archivelog like '/arch%';
```

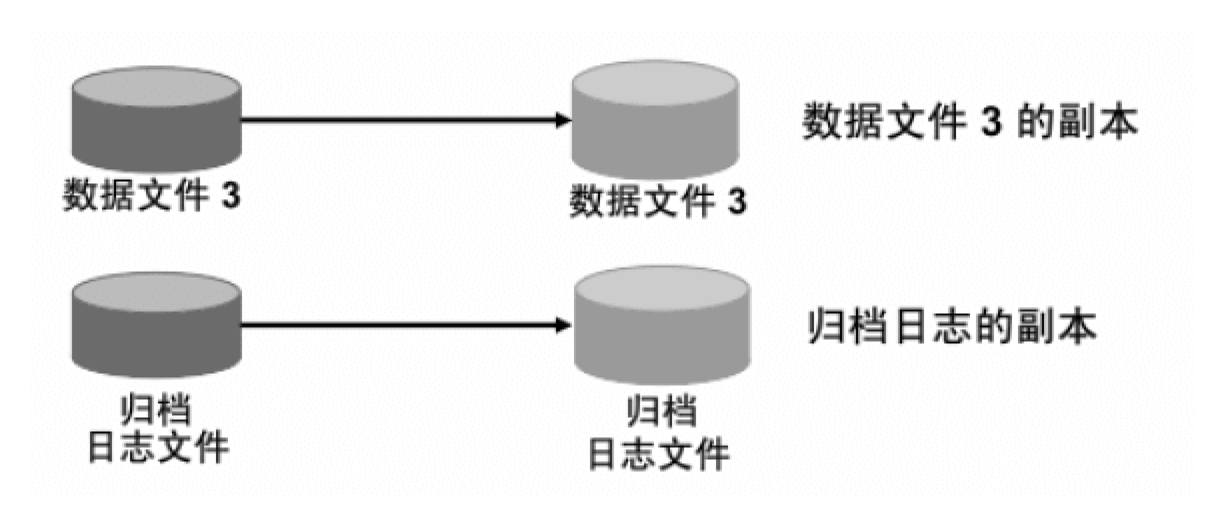


图 6-2　归档日志副本

6.1.2　创建整体数据库备份

整体数据库备份可以是整个数据文件集的备份集或映像副本，其中必须包含控制文件。可以根据需要包括服务器参数文件（SPFILE）和归档重做日志文件，如图 6-3 所示。

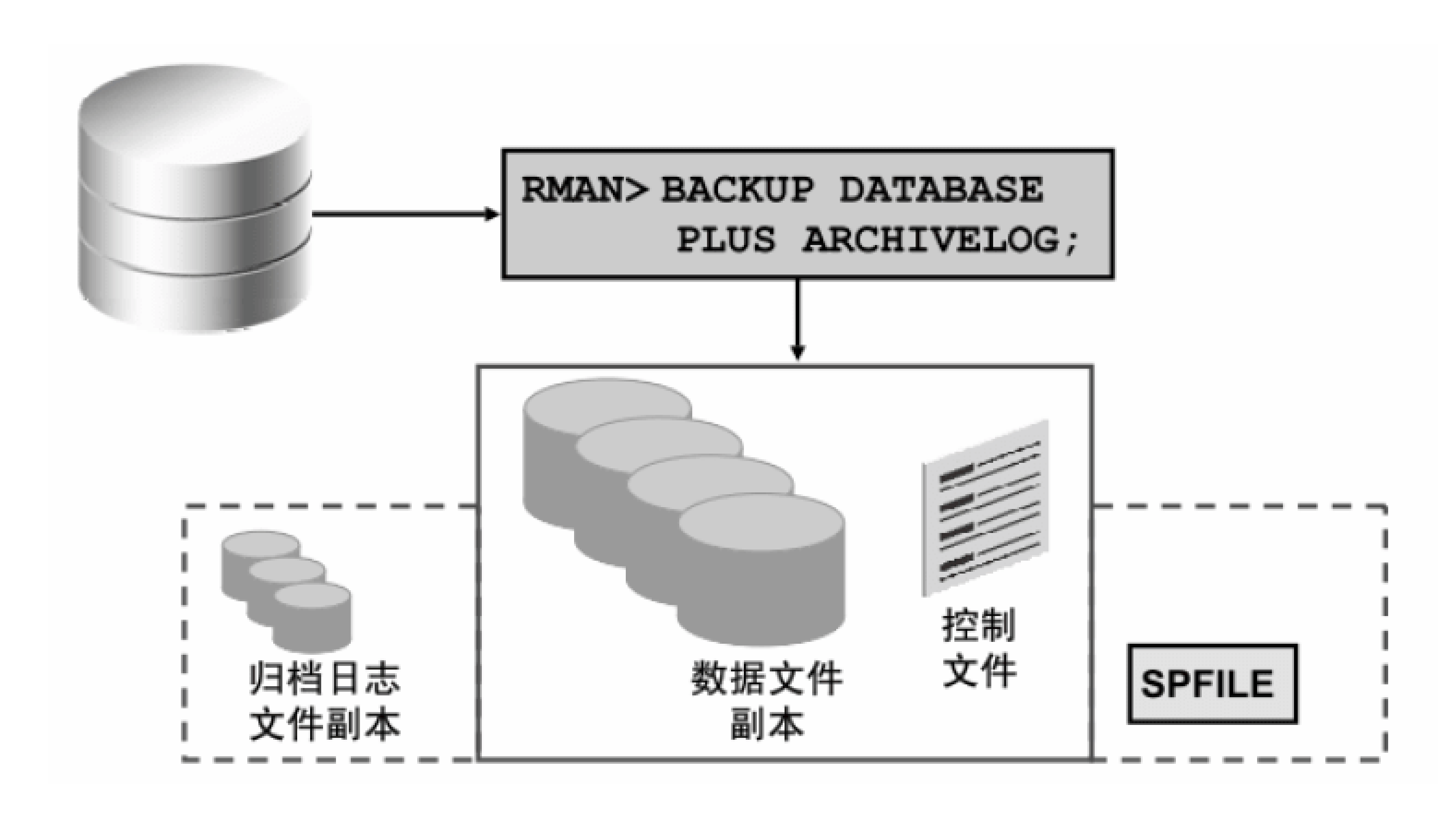


图 6-3　备份文件内容

使用 RecoveryManager（RMAN）可为所有数据库文件创建映像副本，这只需执行以下几步即可完成：装载或打开数据库，启动 RMAN，然后输入 BACKUP 命令。也可以选择在备份归档日志文件时提供 DELETEINPUT 选项。这会使 RMAN 在备份归档日志文件之后将其删除。如果未使用快速恢复区，则该选项尤为有用，它会在空间变得紧张时删除文件，从而执行空间管理。

【示例 6-3】为所有数据库文件创建映像副本

```
rman> backup database plus archivelog delete input;
```

必须在发出以下 CONFIGURE 命令后，才能按上述方式进行备份。

【示例 6-4】CONFIGURE 命令

```
configure default device type to disk;
configure device type disk backup type to copy;
configure controlfile autobackup on;
```

此外还可使用以下命令，为数据库中所有数据文件和控制文件以前的映像副本创建备份（备份集或映像副本）：

【示例 6-5】为映像副本创建备份

```
rman> backup copy of database;
```

默认情况下，RMAN 依次执行每个 BACKUP 命令，也可采用以下方法并行执行复制操作：

- 使用 CONFIGURE DEVICE TYPE DISK PARALLELISM n 命令，其中 n 是所需的并行度。
- 分配多个通道。
- 指定一个 BACKUP AS COPY 命令并列出多个文件。

6.1.3　归档备份

如果需要在指定时间内保留联机备份，RMAN 通常会假定可能需要在自执行该备份以来到现在之间的任意时间执行时间点恢复，如图 6-4 所示。

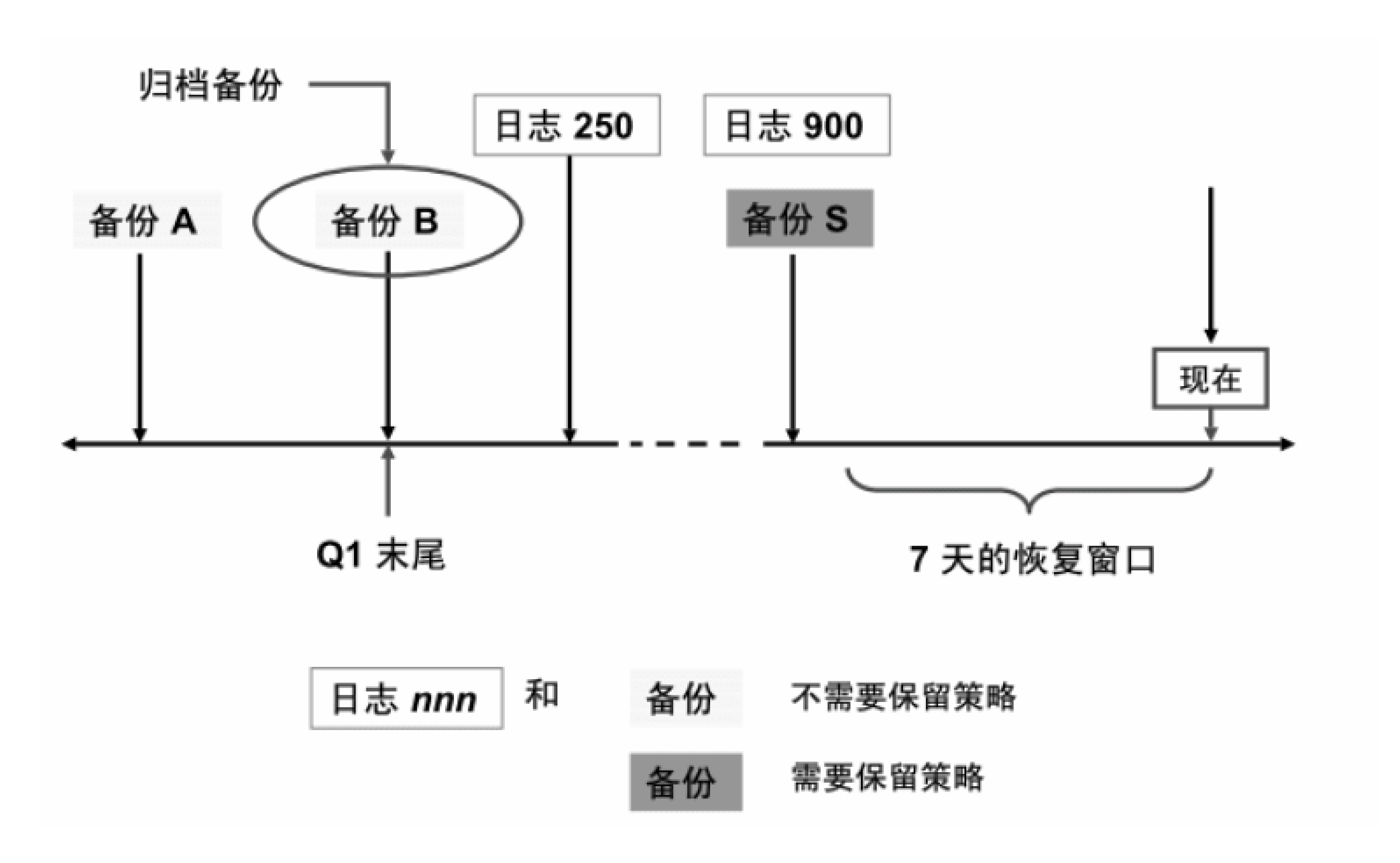


图 6-4　备份策略

为了满足这一要求，RMAN 会在此时段内保留归档日志。但是，可能仅需要在指定的时间（如两年）内保留特定备份（并使其保持一致和可恢复）。不打算恢复到自执行该备份以后的某一时间点，只是希望能够正好恢复到执行该备份的确切时间。此外，还需要维护保留策略以使备份区井然有序，因此无法使备份恢复到两年前。为了满足保留数据的商业或法律要求，通常需要这么做。归档备份可以解决这一问题。如果将某一备份标记为归档备份，该属性将覆盖为此备份目的配置的所有保留策略。保留归档备份时，可将其指定为仅在某一特定时间过时，也可以将其指定为永不过时。如果要指定后者，则需要使用恢复目录。KEEP 子句会创建一个归档备份，此备份是某个时间点的数据库快照。仅保留将此备份还原至一致状态所需的重做日志。在备份完成后发出的 RESTORE POINT 子句可确定所保留的重做日志的数量（足以将备份还原至 RESTORE POINT 时间点的日志数量）。

归档备份还可保证包含还原备份所需的全部文件。RMAN 包含数据文件、归档日志文件（仅限恢复联机备份所需的那些文件）及相关自动备份文件。所有这些文件都必须保存到同一介质系列（或磁带组）中。此外，还可指定要创建的还原点，该还原点与归档备份具有相同的 SCN。实际上，这为执行备份的时间点提供了一个有意义的名称。创建归档备份之后，它将保留指定的时间。即使具有非常短的保留窗口并运行了 DELETE OBSOLETE 命令，归档备份也会保留下来。此备份是数据库在某个时间点的快照，可用于将数据库还原到另一个主机（例如，用于测试目的）。



归档备份不能写入快速恢复区。因此，如果具有快速恢复区，则必须通过 FORMAT 子句指定其他位置。

6.1.4 RMAN 备份类型

在 RMAN 增量备份中，有差异增量和累积增量的概念，如图 6-5 所示。

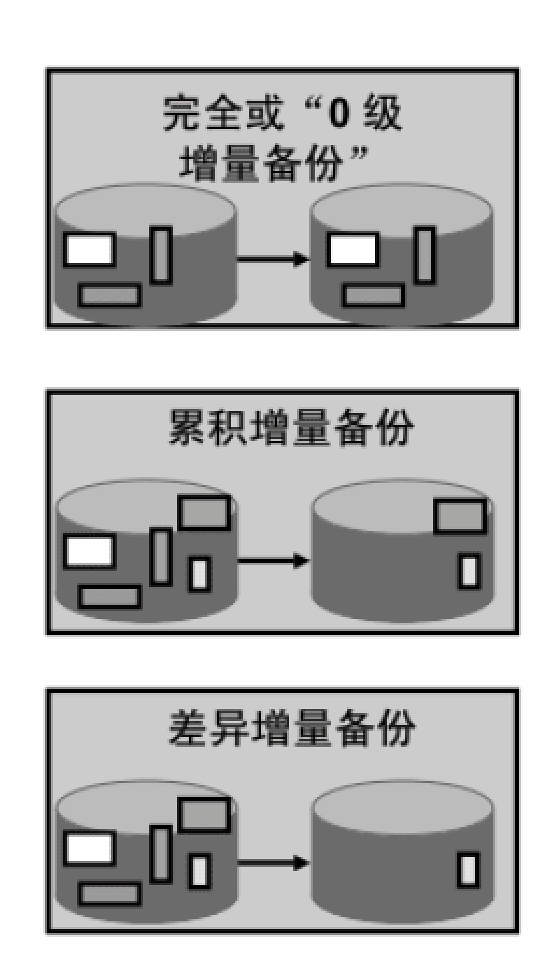


图 6-5　不同的备份设置

（1）差异增量：是备份上级及同级备份以来所有变化的数据块，是默认增量备份方式。

（2）累积增量：是备份上级备份以来所有变化的数据块。因为累积增量是备份上级备份以来所有变化的数据块，所以累积增量需要更多的备份时间，同时需要较小的恢复时间；而差异增量正好相反，它可以备份同级备份以来变化的数据块。所以会减少备份时间，但需要更多的恢复时间。在 Oracle 9 中增量备份需要通过扫描整个数据库的数据块才能知道哪些数据块发生了变化，这是一个代价很大、时间很长的过程，而且由于增量备份形成多个不同的备份集，使恢复变得更加不可靠，所以增量备份在版本 9 中仍然不被推荐使用。在 10g 中，增量备份做了很大的改进，不需要再扫描所有数据块就能得知哪些数据块发生变化（块跟踪），从而大大提升备份效率。但这些却以牺牲磁盘 i/o 为代价，所以在 OLTP 事务系统中还得衡量是否愿意以 i/o 为代价来保证安全及高可用性。10g 还支持增量合并，增量备份可支持 7 级增量。

需要说明的是 10g 起 Oracle 官方只推荐使用 level 0 和 level 1 级备份，10g 还能使用 level 2。首先看下 Oracle 官方解释（载至在线文档 Database Backup and Recovery Basics 4.4 节 RMAN Incremental Backups）：

```
A level 1 incremental backup can be either of the following types:
A differential backup,which backs up all blocks changed after the most recent
incremental backup at level 1 or 0
A cumulative backup,which backs up all blocks changed after the most recent
incremental backup at level 0
Incremental backups are differential by default。
```

也就是 differential 是上次备份（不论是 level 0 或者 level 1）至今的变化数据，这个是 level 1 的默认值。cumulative 是从上次 level 0 备份后至今的所有变化数据，也就是说，如果在 level 0 至今，中间如果有若干次增量备份（level 1 的 differential 或者 cumulative ），所有的变化

内容都将保存在这个增量集中，结合图 6-6 理解。

图 6-6　增量备份策略

增量备份的 level 1 是从上次 0 或者 1 至今的变化，而 level 2 是从上次备份至今的增量，无论是 0 或者 1 或者 2。

累积备份的 level 1 是从上次 0 至今的累积变化，而 level 2 是上次 level 0 或者 level 1 至今的累积增量（包括期间的 level 2 增量累计），如图 6-7 所示。

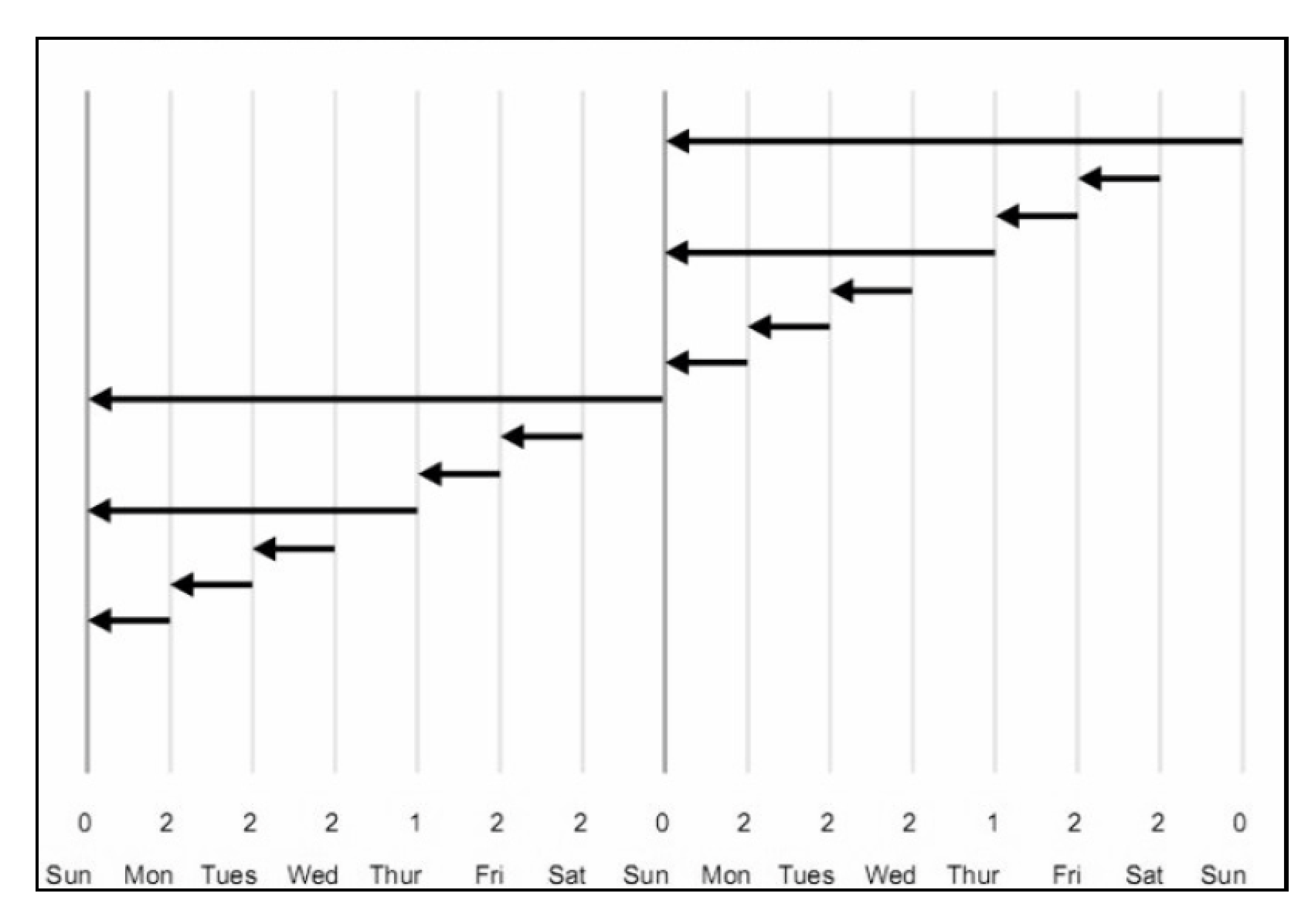


图 6-7　累积备份策略

这里看一个具体例子（见图 6-8）：对一个星期做如下备份策略，如果要恢复到星期五那天，那么差异增量只需要使用星期天的 0 级、星期三的 1 级以及星期四和星期五的 2 级就可以恢复；累积增量则需要星期天的 0 级、星期三的 1 级以及星期四和星期五的 2 级进行恢复。

星期	差异增量	累积增量
星期天	0级	0级
星期一	2级	2级
星期二	2级	2级
星期三	1级	1级
星期四	2级	2级
星期五	2级	2级
星期六	2级	2级
星期天	0级	0级

图 6-8　备份策略配置举例

在具体操作方面，几种不同的备份级别脚本如示例 6-6 所示。

【示例 6-6】各备份级的不同脚本

```
全库备份：
run{
   allocate channel c1 type disk;
   allocate channel c2 type disk;
   allocate channel c3 type disk;
   backup full tag 'dbfull' format '/u01/oradata/backup/full%u_%s_%p' Database
   include current controlfile;
   sql 'alter system archive log current';
   backup filesperset 3 format '/u01/oradata/backup/arch%u_%s_%p'
   archivelog all delete input; #备份归档可选，可以单独定期备份
   release channel c1;
   release channel c2;
   release channel c3;
   }

零级备份：
 run{
   allocate channel c1 type disk;
   allocate channel c2 type disk;
   allocate channel c3 type disk;
   backup incremental level 0 tag 'db0' format '/u01/oradata/backup/db0%u_%s_%p'
   Database skip readonly;
   sql 'alter system archive log current';
   backup filesperset 3 format '/u01/oradata/backup/arch%u_%s_%p'
   archivelog all delete input; #备份归档可选，可以单独定期备份
   release channel c1;
   release channel c2;
   release channel c3;
   }
```

```
1 级备份：
run{
   allocate channel c1 type disk;
   allocate channel c2 type disk;
   allocate channel c3 type disk;
   backup incremental level 1 tag 'db1' format '/u01/oradata/backup/db1%u_%s_%p'
   Database skip readonly;
   sql 'alter system archive log current';
   backup filesperset 3 format '/u01/oradata/backup/arch%u_%s_%p'
   archivelog all delete input; #备份归档可选，可以单独定期备份
   release channel c1;
   release channel c2;
   release channel c3;
   }
```

6.1.5　压缩备份和加密备份

1. 压缩备份

压缩未使用的块可减少写入备份的块数，而二进制压缩可使用算法压缩写入的数据。有两种可用的压缩算法：ZLIB 和 BZIP2。ZLIB 进行了优化，可以提高 CPU 使用效率，而 BZIP2 也经过优化，可使压缩比率达到最大。BZIP2 消耗的 CPU 资源虽然比 ZLIB 多，但是通常会生成更紧密的备份。对于 ZLIB 压缩，必须将 COMPATIBLE 初始化参数设置为 11.0.0 或更高，这需要使用“Oracle Advanced Compression（Oracle 高级压缩）”选项。无须执行任何其他步骤便可还原压缩的备份。应该注意的是，压缩和解压缩操作需要占用 CPU 资源。因此，创建和还原压缩的备份可能需要更长的时间并需要占用更多的系统资源。选择算法时，应考虑磁盘空间以及动态系统资源（如 CPU 和内存等）。

2. 加密备份

使用以下三种方法之一对备份进行加密即可：

- 透明加密：此方法使用 wallet，是默认模式。
- 口令加密：此加密方法依赖于口令，不需要配置 wallet。若要还原，必须知道用于备份的口令。
- 双模式加密：可同时使用透明加密和口令加密。若要还原，可使用透明模式或口令模式。如果通常将备份还原到本地站点，但偶尔也会将备份传送到其他站点，则这种类型的加密十分有用。使用 SET ENCRYPTION 修改加密设置。

【示例 6-7】口令加密

```
rman> set encryption identified by mypassword;
rman> backup datafile 5;
...
rman> set decryption identified by mypassword;
```

RMAN 可透明地加密写入备份集的数据，并可在 RESTORE 操作需要这些数据时解密备份。要在磁盘上创建加密备份，数据库必须使用高级安全选项。要直接在磁带上创建加密备份，RMAN 必须使用 Oracle Secure Backup SBT 接口，但不需要使用高级安全选项。如果要使用透明加密，则必须创建 wallet，并且必须将数据库配置成使用 wallet。透明数据加密也使用同一 wallet。必须先创建 wallet，之后才能将其用于加密的备份或 TDE。

【示例 6-8】向 SQLNET.ORA 文件添加一个条目

```
encryption_wallet_location =
(source =
(method = file)
(method data =
(directory =
/oracle/dbsid/admin/pdcs11/wallet)))
```

【示例 6-9】创建一个简单的受口令保护的 wallet

```
alter system set [encryption] key identified by "welcome1";
```



可以配置为 Oracle Secure Backup（OSB）加密，以便无论客户机请求什么类型的加密，都始终使用 OSB 加密所有备份。

6.2 使用 RMAN 执行恢复

RMAN 的恢复过程主要涉及两个命令：RESTORE 和 RECOVER。

【示例 6-10】RESTORE 和 RECOVER 举例

```
rman> sql 'alter tablespace inv_tbs offline immediate';
rman> restore tablespace inv tbs;
rman> recover tablespace inv tbs;
rman> sql 'alter tablespace inv_tbs online';
```

从备份中重建整个数据库或数据库某一部分的过程通常包含两个阶段：从备份中检索数据文件的副本，以及从归档和联机重做日志中重新应用自备份以来对文件所做的更改，以使数据库恢复到所需的 SCN（通常为最新的 SCN）。

RESTORE 命令将数据文件从磁带、磁盘或其他介质上的备份位置检索到磁盘上，并使其可供数据库服务器使用。RMAN 会从备份中还原恢复操作期间所需的任何归档重做日志。如果备份存储在介质管理器上，则必须配置或分配用于访问存储在介质上的备份的通道。RECOVER 命令获取已还原的数据文件副本，并将数据库重做日志中记录的更改应用于该副本。

6.2.1　完全恢复与不完全恢复

顾名思义，完全恢复就是指数据没有丢失的恢复，不完全恢复就是指恢复后有部分数据丢失。它们是数据库的两种恢复方式。

（1）完全恢复：利用重做日志或增量备份将数据块恢复到最接近当前时间的时间点。之所以叫作完整恢复是由于 Oracle 应用了归档日志和联机重做日志中所有的修改。

（2）不完全恢复：利用备份产生一个非当前版本的数据库。换句话说，恢复过程中不会应用备份产生后生成的所有重做日志。

通常在下列情况下生成整个数据库的不完整恢复。

- 介质失败损坏了几个或全部的联机重做日志文件。
- 用户操作造成的数据丢失，比如用户误删除了一张表。
- 由于个别归档日志文件的丢失无法进行完整的恢复。
- 丢失了当前的控制文件，必须使用备份的控制文件打开数据库。

为了执行不完整介质恢复，必须使用恢复时间点以前的备份来还原数据文件，并在恢复完成后使用 RESETLOG 选项打开数据库 resetlogs 参数。

在不完全恢复期间，通常需要使用 resetlogs 命令打开数据库。这是因为要从已经建立的现有重做日志流中脱离出来，resetlogs 参数表示一个数据库逻辑生存期结束、另一个数据库逻辑生存期开始。数据库的逻辑生存期也称为一个对应物（incarnation）。每次使用 resetlogs 命令时，SCN 计数器不会被重置，但是 Oracle 会重置其他计数器（如日志序列号），同时还会重置联机重做日志的内容。

【示例 6-11】当在非归档模式下尝试执行完全恢复时，发出 alter Database open 后，RMAN 报错

```
sql> alter database open;
alter database open
*
error at line 1:
ora-01589: must use resetlogs or noresetlogs option for database open
```

如果加上 resetlogs 参数，就可避免该报错产生。而且从恢复的过程来看，其间确实使用在联机日志文档，也就是说它执行的是完全恢复。这说明在非归档模式下执行完全恢复后，打开数据库时也要重置重做日志。其实这也很好想，非归档模式下，没有归档的重做日志，完全恢复时使用联机日志后，这些联机日志文件就没什么用了，因此 Oracle 就重置日志文件序列号。（理论上来说，是可以不重置的，日志文件的序号直接在现有的日志序号上增加，但是这样日志序号会越来越大。Oracle 应该是考虑到这一点就在非归档模式下执行完全恢复和不完全恢复时都重置了重做日志。）完全恢复比较简单，如示例 6-12 所示。

【示例 6-12】RMAN 完全恢复的操作命令

```
$ rman target /
```

```
rman> startup mount
rman> restore database;
rman> recover database;
rman> alter database open [resetlogs];
```

【示例 6-13】RMAN 不完全恢复的主要操作命令

```
a、基于 TIME 参数不完全恢复
run {
     shutdown immediate;
     startup mount;
     set until time "to_date('20130705 10:09:53', 'yyyymmdd hh24:mi:ss')";
     restore Database;
     recover Database;
     alter Database open resetlogs;
}

b、基于 SCN 参数不完全恢复
run {
     shutdown immediate;
     startup mount;
     set until scn 3400;
     restore Database;
     recover Database;
     alter Database open resetlogs;
}

c、基于 SEQUENCE 参数不完全恢复
run {
     shutdown immediate;
     startup mount;
     set until sequence 12903;
     restore Database;
     recover Database;
     alter Database open resetlogs;
}
```

相比与完全恢复，Oracle 的不完全恢复显得较复杂，并且应用场景也较多，其特点主要体现在以下几方面：

- RMAN 支持基于 TIME、SCN、SEQUENCE 参数的不完全恢复，不支持基于 CANCEL 的不完全恢复。
- 所有实施了不完全恢复的数据库都需要以 open resetlogs 方式打开数据库，且同时伴随一个新的 incarnation 产生。
- 不完全恢复之后即使是恢复到故障点，或者说想做完全恢复，都只能是做不完全恢复到最近时刻。
- 不完全恢复后再次恢复到最新时刻，新的 incarnation 变为 CURRENT 状态，中间的 incarnation 为 ORPHAN 状态。

- 首次不完全恢复以 open resetlogs 方式打开数据库时，未归档的联机日志被归档。
- until 子句是一个非半闭包的形式。
- 生产环境建议不完全恢复前后备份数据库。

6.2.2 RMAN 演示 1：在丢失了所有控制文件副本后进行恢复

正常情况下，Oracle 数据库的日常备份是要求对重要的控制文件进行备份的，当发现控制文件的所有副本均已丢失时，可以通过 RMAN 执行恢复操作。

删除数据库当前正在使用的控制文件的所有副本。

（1）在 SQL*Plus 会话中，列出控制文件。

【示例 6-14】列出控制文件

```
SQL> show parameter control files
NAME                                 TYPE        VALUE
------------------------------------ ----------- -------------
control_files                          string
+DATA/orcl/controlfile/current
.256.628901483,  +FRA/orcl/cont
rolfile/current.256.628901483
SQL>
```

（2）关闭 ORCL 实例，以便可以删除控制文件。

【示例 6-15】关闭实例

```
SQL> shutdown immediate
Database closed.
Database dismounted.
ORACLE instance shut down.
SQL>
```

（3）运行 rm_asm_file.sh 脚本，删除每个控制文件。

【示例 6-16】删除控制文件

```
SQL> host ./rm_asm_file.sh
+DATA/orcl/controlfile/current.256.628901483
ASMCMD> ASMCMD>
SQL> host ./rm_asm_file.sh
+FRA/orcl/controlfile/current.256.628901483
ASMCMD> ASMCMD>
```

（4）启动数据库。查看此时数据库处于 NOMOUNT 状态，无法成功启动数据库，提示报错信息 ORA-00205。

【示例 6-17】启动数据库

```
SQL> startup
ORACLE instance started.
```

```
Total System Global Area  389189632 bytes
Fixed Size                 1300128 bytes
Variable Size            356518240 bytes
Database Buffers          25165824 bytes
Redo Buffers               6205440 bytes
ORA-00205: error in identifying control file,  check alert log
for more info
SQL> select status from v$instance;
STATUS
------------
STARTED
SQL>
```



仅启动了实例，而没有装载数据库。这是因为控制文件已丢失。

（5）根据 SPFILE 创建初始化参数文件 （pfile）。

【示例 6-18】根据 SPFILE 创建初始化参数文件

```
SQL> create pfile from spfile;
File created.
```

（6）找到并查看 initorcl.ora 文件。

【示例 6-19】查看文件

```
$ vi /u01/app/Oracle/product/11.1.0/db_1/dbs/initorcl.ora &
```

（7）编辑 initorcl.ora 文件，使控制文件设置指向先前创建的映像副本，务必保存文件。

【示例 6-20】编辑文件

```
*.control_files='/tmp/ctl.bak'#Restore Controlfile
```

（8）关闭实例。

【示例 6-21】关闭实例

```
SQL> shutdown abort
```

（9）使用编辑后的参数文件启动实例，该文件指向控制文件副本。

【示例 6-22】启动实例

```
SQL> startup pfile=?/dbs/initorcl.ora
ORACLE instance started.
Total System Global Area  627732480 bytes
Fixed Size                 1301728 bytes
Variable Size            570426144 bytes
Database Buffers          50331648 bytes
Redo Buffers               5672960 bytes
Database mounted.
ORA-01589: must use RESETLOGS or NORESETLOGS option for
```

```
Database open
SQL>
```



不指定 RESETLOGS 将无法打开数据库，因为正在使用备份的控制文件。

（10）使用 RESETLOGS 选项打开数据库。

【示例 6-23】打开数据库

```
SQL> alter Database open resetlogs;
alter Database open resetlogs
*
ERROR at line 1:
ORA-01152: file 1 was not restored from a sufficiently old
backup
ORA-01110: data file 1:
'+DATA/orcl/datafile/system.258.630811685'
```



需要恢复数据库。

（11）通过使用备份控制文件选项来恢复数据库。

【示例 6-24】恢复数据库

```
SQL> recover Database using backup controlfile;
ORA-00279: change 1377484 generated at 08/20/2007 08:07:30
needed for thread 1
ORA-00289: suggestion : +FRA
ORA-00280: change 1377484 for thread 1 is in sequence #46
```

（12）需要先输入 CANCEL 执行必要的查询，以确定要应用的日志文件的名称。

【示例 6-25】查询日志文件名称

```
Specify log: {<RET>=suggested | filename | AUTO | CANCEL}
CANCEL
```

（13）先查询归档重做日志文件的信息，确定与给定序列号关联的重做日志。



序列号可能与示例中的序列号不同。

【示例 6-26】查询归档重做日志文件的信息

```
SQL> select * from v$archived_log where sequence# = 46;
no rows selected
SQL>
```

如果未在 V$ARCHIVED_LOG 视图中找到该序列号，则该序列号没有对应的归档文件。

（14）请检查该序列号对应的联机重做日志文件。

【示例 6-27】检查对应的联机重做日志文件

```
SQL> select * from v$log where sequence# = 46;
  GROUP#    THREAD#   SEQUENCE#    BYTES  MEMBERS  ARC  STATUS
--------  ---------  ----------   ------ --------  ---  ------
FIRST_CHANGE# FIRST_TIME
------------- -------------------
2         1          46 52428800       2   NO  CURRENT
     1376170 2007-08-20:08:07:30
```



查看关联的联机重做文件的 group#编号。查询 V$LOG 视图，确定该序列号对应的文件名。组编号可能与此处显示的组编号不同。

【示例 6-28】查询 V$LOG 视图

```
SQL> select * from v$logfile where group#=2;
GROUP#  STATUS  TYPE
----------  ------- -------
MEMBER
--------------------------------------------------------------
------------------
IS_
---
       2        ONLINE
+DATA/orcl/onlinelog/group_2.268.630812389
NO
       2        ONLINE
+FRA/orcl/onlinelog/group_2.259.630812413
YES
```



该组的文件名。重新尝试执行恢复命令，在系统提示时提供其中的一个文件名。提供的文件名很可能与示例中显示的文件名不同。

【示例 6-29】执行恢复命令

```
SQL> recover Database using backup controlfile;
ORA-00279: change 1377484 generated at 08/20/2007 08:07:30
needed for thread 1
ORA-00289: suggestion : +FRA
ORA-00280: change 1377484 for thread 1 is in sequence #46
Specify log: {<RET>=suggested | filename | AUTO | CANCEL}
+DATA/orcl/onlinelog/group_2.268.630812389
Log applied.
Media recovery complete.
SQL>
```

（15）使用 RESETLOGS 选项重新尝试打开数据库。

【示例 6-30】重新打开数据库

```
SQL> alter Database open resetlogs;
Database altered.
SQL>
```

（16）数据库已成功打开，但是使用 pfile 打开的，切换回使用 spfile。需要创建 spfile，然后重启数据库，在没有指定 pfile 的情况下启动数据库，将默认使用 spfile 进行启动。

【示例 6-31】创建 spfile

```
SQL> create spfile from pfile;
File created
SQL>
```

（17）关闭数据库。

【示例 6-32】关闭数据库

```
SQL> shutdown immediate
Database closed.
Database dismounted.
ORACLE instance shut down.
SQL>
```

（18）启动数据库，然后退出 SQL*Plus。

【示例 6-33】退出 SQL*Plus

```
SQL> startup
ORACLE instance started.
Total System Global Area  627732480 bytes
Fixed Size                  1301728 bytes
Variable Size             570426144 bytes
Database Buffers           50331648 bytes
Redo Buffers                5672960 bytes
Database mounted.
Database opened.
SQL> exit
```

6.2.3　RMAN 演示 2：在丢失了重做日志组后进行恢复

（1）关闭数据库，以便可以删除文件，然后退出 SQL*Plus。

【示例 6-34】关闭数据库

```
$ sqplus / as sysdba
SQL> shutdown immediate
SQL> exit
```

（2）使用 rm_asm_file.sh 脚本删除当前组中的两个重做日志文件。

【示例 6-35】删除当前组中的两个重做日志文件

```
$ ./rm asm file.sh +FRA/orcl/onlinelog/group 4.258.629772261
$ ./rm_asm_file.sh +DATA/orcl/onlinelog/group_4.265.629772243
```

（3）启动数据库。

【示例 6-36】启动数据库

```
SQL> startup
ORACLE instance started.
Total System Global Area  422670336 bytes
Fixed Size                 1300352 bytes
Variable Size            348129408 bytes
Database Buffers          67108864 bytes
Redo Buffers               6131712 bytes
Database mounted.
ORA-00313: open failed for members of log group 4 of thread 1
ORA-00312: online log 4 thread 1:
'+DATA/orcl/onlinelog/group 4.265.629772243'
ORA-00312: online log 4 thread 1:
'+FRA/orcl/onlinelog/group_4.258629772261'
```



数据库未打开。请查看有关丢失联机重做日志文件的错误消息。同时请注意，数据库未处于 MOUNT 状态。在本例中，相关的组编号为 4。

（4）如果当前日志组丢失，则无法进行完全恢复。只可能恢复到与当前最接近的时点。

（5）清除日志文件。

【示例 6-37】清除日志文件

```
SQL> alter Database clear logfile group 4;
alter Database clear logfile group 4
*
ERROR at line 1:
ORA-00350: log 4 of instance orcl (thread 1) needs to be
archived
ORA-00312: online log 4 thread 1:
'+DATA/orcl/onlinelog/group 4.265.629772243'

ORA-00312: online log 4 thread 1:
'+FRA/orcl/onlinelog/group_4.258.629772261'
SQL>
```



无法清除当前的重做日志组，因为该日志组未归档。处于当前状态下的日志文件从不归档，必须先将日志文件的状态更改为活动或非活动。

（6）再次尝试清除日志文件组，但这次指明该组未归档。

【示例 6-38】再次尝试清除日志文件组

```
SQL> alter Database clear unarchived logfile group 4;
Database altered.
SQL>
```

（7）打开数据库，然后退出 SQL*Plus。

【示例 6-39】打开数据库

```
SQL> alter Database open;
Database altered.
SQL> exit
```

6.2.4　RMAN 演示 3：恢复映像副本

把文件的映像副本恢复至当前的 SCN，以便提高以后的恢复速度。新建名为 APPRAISAL 的表空间，其中包含一个表。完成初始创建后，在只有少量数据的情况下创建表空间的增量备份。然后，对表空间执行一些 DML 操作，添加大量行。接着创建另一个增量备份。此时，已拥有 APPRAISAL 表空间的映像副本和增量备份。因为以后需要恢复表空间，所以可以恢复映像副本，这样映像副本即具有与上次增量备份一致的最新数据。执行此操作不需要创建新的映像副本。

（1）新建一个名为 APPRAISAL 的表空间。

（2）为该表空间创建一个 1 级备份，用于恢复映像副本。如果不存在 1 级备份，则实际上创建了一个 0 级增量备份。完成此操作需要几分钟时间。

【示例 6-40】创建备份

```
$ rman target / catalog rcatowner/Oracle@rcat
RMAN> backup incremental level 1 for recover of copy with tag
'app_incr' Database;
Starting backup at 05.10.07:23:02:59
new incarnation of Database registered in recovery catalog
starting full resync of recovery catalog
full resync complete
allocated channel: ORA_DISK_1
channel ORA_DISK_1: SID=112 device type=DISK
no parent backup or copy of datafile 1 found
no parent backup or copy of datafile 2 found
no parent backup or copy of datafile 3 found
no parent backup or copy of datafile 5 found
……
Finished backup at 2007-05:23:17:03
Starting Control File and SPFILE Autobackup at 2007-10-
5:23:17:04
piece
handle=+FRA/orcl/autobackup/2007 10 05/s 635210233.297.6352102
37 comment=NONE
```

```
Finished Control File and SPFILE Autobackup at 2007-10-
5:23:17:23
RMAN>
```

（3）在另一个终端窗口中，对 APPRAISAL 表空间中的表执行一些 DML 操作。

（4）列出 APPRAISAL 表空间的副本，查看其 SCN。

【示例 6-41】查看 SCN

```
RMAN> list copy of tablespace appraisal;
List of Datafile Copies
=======================
Key     File S Completion Time     Ckp SCN    Ckp Time
------- ---- - ------------------- --------- ------------------
630     7    A 2007-10-5:23:16:51  822339    2007-10-
05:23:16:45
    Name: +FRA/orcl/datafile/appraisal.295.635210207
    Tag: APP INCR
RMAN>
```

（5）手动对 APPRAISAL 表空间中的表执行一些事务处理。

（6）执行另一个 1 级备份，该备份必然是一个 1 级备份，因为已经有了 0 级备份。

【示例 6-42】执行备份

```
RMAN> backup incremental level 1 for recover of copy with tag
'app_incr' Database;
Starting backup at 05.10.07:23:37:02
allocated channel: ORA_DISK_1
channel ORA_DISK_1: SID=109 device type=DISK
channel ORA DISK 1: starting incremental level 1 datafile
backup set
channel ORA_DISK_1: specifying datafile(s) in backup set
input datafile file number=00001
name=+DATA/asmtesting/datafile/testing.257.635151709
input datafile file number=00002
name=+DATA/orcl/datafile/sysaux.259.635127417
input datafile file number=00003
name=+DATA/orcl/datafile/undotbs1.258.635127417input datafile
file number=00005
name=+DATA/orcl/datafile/example.260.635127419
input datafile file number=00006
name=+DATA/orcl/datafile/tbsjmw.263.635148067
input datafile file number=00007
name=+DATA/orcl/datafile/appraisal.256.635209311
input datafile file number=00004
name=+DATA/orcl/datafile/users.262.635127891
channel ORA_DISK_1: starting piece 1 at 2007-10-05:23:37:11
channel ORA DISK 1: finished piece 1 at 2007-10-05:23:37:36
piece
handle=+FRA/orcl/backupset/2007_10_05/nnndn1_tag20071005t23370
```

```
2_0.298.635211431 tag=TAG20071005T233702 comment=NONE
channel ORA_DISK_1: backup set complete, elapsed time:
00:00:25
Finished backup at 05.10.07:23:37:36
Starting Control File and SPFILE Autobackup at 2007-10-
5:23:37:36
piece
handle=+FRA/orcl/autobackup/2007 10 05/s 635211458.299.6352114
67 comment=NONE
Finished Control File and SPFILE Autobackup at 2007-10-
5:23:37:53
RMAN>
```

（7）列出 APPRAISAL 表空间增量备份（而不是映像副本）的 SCN 并注意其数值。

【示例 6-43】列出 SCN

```
RMAN> list backup of tablespace appraisal;
List of Backup Sets
===================
BS Key  Type LV Size  Device Type Elapsed Time Completion Time
------- ---- -- ----- ----------- ------------ ---------------
676     Incr 1  12.72M  DISK     00:00:24     2007-10-
05:23:37:27
       BP Key: 679   Status: AVAILABLE  Compressed: NO  Tag:
TAG20071005T233702       Piece Name:
+FRA/orcl/backupset/2007_10_05/nnndn1_tag20071005t233702_0.298
.635211431
List of Datafiles in backup set 676
  File LV Type Ckp SCN    Ckp Time            Name
  ---- -- ---- ---------- ------------------- ----
  7    1  Incr 824887     2007-10-05:23:37:11
+DATA/orcl/datafile/appraisal.256.635209311
RMAN>
```

（8）使用增量备份恢复 APPRAISAL 表空间的映像副本。

【示例 6-44】使用增量备份恢复 APPRAISAL 表空间的映像副本

```
RMAN> recover copy of tablespace appraisal with tag
'app_incr';
Starting recover at 2007-10-5:23:52:00
allocated channel: ORA DISK 1
channel ORA_DISK_1: SID=170 device type=DISK
channel ORA_DISK_1: starting incremental datafile backup set
restore
channel ORA_DISK_1: specifying datafile copies to recover
recovering datafile copy file number=00007
name=+FRA/orcl/datafile/appraisal.295 .635210207
channel ORA_DISK_1: reading from backup piece
+FRA/orcl/backupset/2007_10_05/nnn
dn1_tag20071005t233702_0.298.635211431
```

```
channel ORA_DISK_1: piece
handle=+FRA/orcl/backupset/2007_10_05/nnndn1_tag200710
05t233702 0.298.635211431 tag=TAG20071005T233702
channel ORA DISK 1: restored backup piece 1
channel ORA_DISK_1: restore complete， elapsed time: 00:00:01
Finished recover at 2007-10-5:23:52:03
……
RMAN>
```

（9）列出 APPRAISAL 表空间映像副本的 SCN。

【示例 6-45】列出 SCN

```
RMAN>list copy of tablespace appraisal;
List of Datafile Copies
=======================
Key     File S Completion Time     Ckp SCN    Ckp Time
------- ---- - ------------------- ---------- ----------------
732     7    A 2007-10-5:23:52:02  824887     2007-10-
05:23:37:11
       Name: +FRA/orcl/datafile/appraisal.295.635210207
       Tag: APP_INCR
RMAN>
```



它现在等于上次增量备份的 SCN。

（10）根据最新的增量备份恢复数据库中所有数据文件的映像副本。

【示例 6-46】恢复数据库中所有数据文件的映像副本

```
RMAN> recover copy of Database with tag 'app incr';
Starting recover at 2007-10-6:0:00:17
allocated channel: ORA_DISK_1
channel ORA_DISK_1: SID=112 device type=DISK
no copy of datafile 7 found to recover
channel ORA DISK 1: starting incremental datafile backup set
restore
channel ORA_DISK_1: specifying datafile copies to recover
recovering datafile copy file number=00001
name=+FRA/orcl/datafile/system.290.635209395
recovering datafile copy file number=00002
name=+FRA/orcl/datafile/sysaux.291.635209755
recovering datafile copy file number=00003
name=+FRA/orcl/datafile/undotbs1.292.635210025
recovering datafile copy file number=00004
name=+FRA/orcl/datafile/users.296.635210221
recovering datafile copy file number=00005
name=+FRA/orcl/datafile/example.293.635210123
recovering datafile copy file number=00006
name=+FRA/orcl/datafile/tbsjmw.294.635210175
channel ORA_DISK_1: reading from backup piece
```

```
+FRA/orcl/backupset/2007_10_05/nnndn1_tag20071005t233702_0.298
.635211431
channel ORA DISK 1: piece
handle=+FRA/orcl/backupset/2007 10 05/nnndn1 tag20071005t23370
2_0.298.635211431 tag=TAG20071005T233702
channel ORA_DISK_1: restored backup piece 1
channel ORA_DISK_1: restore complete, elapsed time: 00:00:26
Finished recover at 06.10.07:00:00:54
……
RMAN>
```

（11）现在查看所有映像副本的 SCN。这些 SCN 应该都是最新的，等于最近一次增量备份的 SCN。

【示例 6-47】查看所有映像副本的 SCN

```
RMAN> list copy;
List of Datafile Copies
=======================
Key     File S Completion Time    Ckp SCN    Ckp Time
------- ---- - ------------------ ---------- ----------------
800     1    A 06.10.07:0:00:39   824887    05.10.07:23:37:11
        Name: +FRA/orcl/datafile/system.290.635209395
        Tag: APP INCR
802     2    A 06.10.07:0:00:46 824887    05.10.07:23:37:11

Name: +FRA/orcl/datafile/sysaux.291.635209755
        Tag: APP_INCR
801     3    A 2007-10-06:00:00:40 824887    2007-10-
05:23:37:11
        Name: +FRA/orcl/datafile/undotbs1.292.635210025
        Tag: APP_INCR
799     4    A 2007-10-06:00:00:34 824887    2007-10-
05:23:37:11
        Name: +FRA/orcl/datafile/users.296.635210221
        Tag: APP INCR
506     4    A 2007-10-05:05:18:05 799678    2007-10-
05:05:18:03
        Name: +FRA/orcl/datafile/users.279.635145483
        Tag: TAG20071005T051803
414     4    A 2007-10-05:04:53:32 797391    2007-10-
05:04:53:31
        Keep: BACKUP_LOGS       Until: FOREVER
        Name: /tmp/bu_ORCL_24_1.dbf
        Tag: TAG20071005T045326
797     5    A 2007-10-06:00:00:32 824887    2007-10-
05:23:37:11
        Name: +FRA/orcl/datafile/example.293.635210123
        Tag: APP_INCR
798     6    A 2007-10-06:00:00:33 824887    2007-10-
05:23:37:11
```

```
       Name: +FRA/orcl/datafile/tbsjmw.294.635210175
       Tag: APP_INCR
732    7    A 2007-10-05:23:52:02 824887   2007-10-
05:23:37:11
       Name: +FRA/orcl/datafile/appraisal.295.635210207
       Tag: APP_INCR
……
RMAN>
```

6.2.5 RMAN 演示 4：执行快速恢复

利用快速恢复区对数据文件执行快速恢复。

（1）使用上一小节中的 RMAN 会话，并使 APPRAISAL 数据文件脱机。

【示例 6-48】使 APPRAISAL 数据文件脱机

```
RMAN> sql "alter tablespace appraisal offline";
sql statement: alter tablespace appraisal offline
starting full resync of recovery catalog
full resync complete
RMAN>
```

（2）使用 SWITCH 命令替换快速恢复区中的数据文件。

（3）确定当前与 APPRAISAL 表空间关联的数据文件的名称。

【示例 6-49】确定当前与 APPRAISAL 表空间关联的数据文件的名称

```
RMAN> report schema;
Report of Database schema for Database with db_unique_name
ORCL
List of Permanent Datafiles
===========================
File Size(MB) Tablespace          RB segs Datafile Name
---- -------- -------------------- ------- ------------------
1    700      SYSTEM              YES
+DATA/asmtesting/datafile/testing.258.634268145
2    679      SYSAUX              NO
+DATA/orcl/datafile/sysaux.259.634164339
3    200      UNDOTBS1            YES
+DATA/orcl/datafile/undotbs1.260.634164355
4    5        USERS               NO
+DATA/orcl/datafile/users.263.634165219
5    100      EXAMPLE             NO
+DATA/orcl/datafile/example.261.634164377
6    50       TBSJMW              NO
+DATA/orcl/datafile/tbsjmw.269.634202579
7    25       APPRAISAL           NO
+DATA/orcl/datafile/appraisal.256.634276429
.
.
```

```
.
RMAN>
```



表空间只有一个数据文件。在本例中，数据文件编号为 7。可以在接下来的一组命令中使用该数据文件编号，而不是表空间名称。

（4）确认数据文件 7 有可用于切换的映像副本。

【示例 6-50】确认信息

```
RMAN> list copy of datafile 7;
List of Datafile Copies
=======================
Key     File S Completion Time     Ckp SCN    Ckp Time
------- ---- - ------------------- ---------- ----------------
832     7    A 2007-09-26:4:19:58  1287901    2007-09-
26:04:18:13
       Name: +FRA/orcl/datafile/appraisal.295.634277277
       Tag: APP_INCR
RMAN>
```

（5）切换到该映像文件副本。

【示例 6-51】切换副本

```
RMAN> switch datafile 7 to copy;
datafile 7 switched to datafile copy
"+FRA/orcl/datafile/appraisal.295.634277277"
starting full resync of recovery catalog
full resync complete
RMAN>
```

（6）恢复数据文件 7。

【示例 6-52】恢复文件

```
RMAN> recover datafile 7;
Starting recover at 2007-09-26:04:40:37
using channel ORA_DISK_1
starting media recovery
media recovery complete, elapsed time: 00:00:01
Finished recover at 2007-09-26:04:40:38
RMAN>
```

（7）使 APPRAISAL 表空间重新联机。

【示例 6-53】使 APPRAISAL 表空间重新联机

```
RMAN> sql "alter tablespace appraisal online";
sql statement: alter tablespace appraisal online
starting full resync of recovery catalog
full resync complete
```

```
RMAN>
```

（8）生成方案的报表，注意 APPRAISAL 表空间的文件名称。

【示例 6-54】生成方案的报表

```
RMAN> report schema;
Report of Database schema for Database with db unique name
ORCL
List of Permanent Datafiles
===========================
File Size(MB) Tablespace          RB segs Datafile Name
---- -------- -------------------- ------- -------------------
-----
1    700      SYSTEM              YES
+DATA/asmtesting/datafile/testing.258.634268145
2    679      SYSAUX              NO
+DATA/orcl/datafile/sysaux.259.634164339
3    200      UNDOTBS1            YES
+DATA/orcl/datafile/undotbs1.260.634164355
4    5        USERS               NO
+DATA/orcl/datafile/users.263.634165219
5    100      EXAMPLE             NO
+DATA/orcl/datafile/example.261.634164377
6    50       TBSJMW              NO
+DATA/orcl/datafile/tbsjmw.269.634202579
7    25       APPRAISAL           NO
+FRA/orcl/datafile/appraisal.295.634277277
```



现在将快速恢复区数据文件用作联机表空间的打开数据文件。

（9）确定原始数据文件发生了什么情况。

列出数据文件 7 的映像副本，查看其是否列出。

【示例 6-55】列出数据文件 7 的映像副本

```
RMAN> list copy of datafile 7;
List of Datafile Copies
=======================
Key     File S Completion Time     Ckp SCN    Ckp Time
------- ---- - ------------------- ---------- ----------------
982     7    A 2007-09-26:04:37:21 1288660    2007-09-
26:04:25:52
        Name: +DATA/orcl/datafile/appraisal.256.634276429
```



原始数据文件仍然存在，并且作为映像副本列出。但是，切换的原因是该数据文件被认为位于损坏的磁盘上，或者该数据文件本身已受损。

（10）由于切换的原因是该数据文件被认为位于损坏的磁盘上，或者该数据文件本身已在

某种程度上受损，因此运行 rm_asm_file.sh 脚本删除该文件。

【示例 6-56】删除文件

```
$ ./rm_asm_file.sh +DATA/orcl/datafile/appraisal.256.634276429
```

（11）最终，要避免使用快速恢复区作为活动数据文件的存储位置。现在，硬件或损坏问题已得到解决，所以应将数据文件切换回原始位置。

（12）备份数据文件，指定 DATA ASM 磁盘组并使用 DATAFILE 模板。

【示例 6-57】备份数据文件

```
RMAN> backup as copy to destination '+DATA(datafile)' datafile 7;
Starting backup at 2007-09-26:05:11:02
using channel ORA_DISK_1
channel ORA DISK 1: starting datafile copy
input datafile file number=00007
name=+FRA/orcl/datafile/appraisal.295.634277277
output file name=+DATA/orcl/datafile/appraisal.256.634281073
tag=TAG20070926T051102 RECID=33 STAMP=634281075
channel ORA_DISK_1: datafile copy complete, elapsed time: 00:00:03
Finished backup at 26.09.07:05:11:15
Starting Control File and SPFILE Autobackup at 2007-09-26:05:11:16 piece
handle=+FRA/orcl/autobackup/2007_09_26/s_634281078.304.634281091 comment=NONE
Finished Control File and SPFILE Autobackup at 2007-09-26:05:11:33
RMAN>
```

（13）使数据文件脱机。

【示例 6-58】使数据文件脱机

```
RMAN> sql "alter Database datafile 7 offline";
sql statement: alter Database datafile 7 offline
RMAN>
```

（14）将数据文件切换至新创建的副本。

【示例 6-59】将数据文件切换至新创建的副本

```
RMAN> switch datafile 7 to copy;
datafile 7 switched to datafile copy
"+DATA/orcl/datafile/appraisal.256.634281073"
starting full resync of recovery catalog
full resync complete
RMAN>
d) 生成方案的报表，确定数据文件的位置已更改.
RMAN> report schema;
Report of Database schema for Database with db_unique_name
ORCL
List of Permanent Datafiles
===========================
```

```
File Size(MB) Tablespace          RB segs Datafile Name
---- -------- -------------------- -------- -------------------
1    700      SYSTEM              YES
+DATA/asmtesting/datafile/testing.258.634268145
2    679      SYSAUX              NO
+DATA/orcl/datafile/sysaux.259.634164339
3    200      UNDOTBS1            YES
+DATA/orcl/datafile/undotbs1.260.634164355
4    5        USERS               NO
+DATA/orcl/datafile/users.263.634165219
5    100      EXAMPLE             NO
+DATA/orcl/datafile/example.261.634164377
6    50       TBSJMW              NO
+DATA/orcl/datafile/tbsjmw.269.634202579
7    25       APPRAISAL           NO
+DATA/orcl/datafile/appraisal.256.634281073
```

（15）恢复数据文件。

【示例 6-60】恢复数据文件

```
RMAN> recover datafile 7;
Starting recover at 2007-09-26:05:14:48
using channel ORA_DISK_1
starting media recovery
media recovery complete, elapsed time: 00:00:00
Finished recover at 2007-09-26:05:14:49
RMAN>
```

（16）使数据文件联机。

【示例 6-61】使数据文件联机

```
RMAN> sql "alter Database datafile 7 online";
sql statement: alter Database datafile 7 online
RMAN>
```

（17)交叉检查映像副本备份并删除数据文件 7 的过时映像副本。完成操作后退出 RMAN。

【示例 6-62】交叉检查映像副本备份并删除数据文件 7 的过时映像副本

```
RMAN> crosscheck copy;
released channel: ORA DISK 1
allocated channel: ORA_DISK_1
channel ORA_DISK_1: SID=118 device type=DISK
validation succeeded for datafile copy
datafile copy file
name=+FRA/orcl/datafile/system.290.634276557 RECID=30
STAMP=634278061
validation succeeded for datafile copy
……
datafile copy file
name=+FRA/orcl/datafile/appraisal.295.634277277 RECID=34
```

```
STAMP=634281188
validation failed for datafile copy
datafile copy file
name=+DATA/orcl/datafile/appraisal.256.634276429 RECID=32
STAMP=634279041
validation succeeded for archived log
archived log file
name=/u01/app/Oracle/product/11.1.0/db 1/dbs/arch1 36 63124294
3.dbf RECID=7 STAMP=634245437
validation succeeded for archived log
archived log file
name=+FRA/orcl/archivelog/2007_09_25/thread_1_seq_36.279.63424
5437 RECID=8 STAMP=634245437
validation succeeded for archived log
……
Crosschecked 28 objects
RMAN> list expired copy;
specification does not match any archived log in the recovery
catalog
List of Datafile Copies
=======================
Key     File S Completion Time     Ckp SCN    Ckp Time
------- ---- - ------------------- ---------- ----------------
982     7    X 2007-09-26:04:37:21 1288660    2007-09-
26:04:25:52
Name:+DATA/orcl/datafile/appraisal.256.634276429
RMAN> delete expired copy;
released channel: ORA_DISK_1
allocated channel: ORA_DISK_1
channel ORA_DISK_1: SID=118 device type=DISK
specification does not match any archived log in the recovery
catalog
List of Datafile Copies
=======================
Key     File S Completion Time     Ckp SCN    Ckp Time
------- ---- - ------------------- ---------- ----------------
982     7    X 2007-09-26:04:37:21 1288660    2007-09-
26:04:25:52
       Name: +DATA/orcl/datafile/appraisal.256.634276429
Do you really want to delete the above objects (enter YES or
NO)? yes
deleted datafile copy
datafile copy file
name=+DATA/orcl/datafile/appraisal.256.634276429 RECID=32
STAMP=634279041
Deleted 1 EXPIRED objects
RMAN> exit
```

6.2.6 RMAN 演示 5：克隆数据库

Oracle 数据库克隆也叫作 Oracle 数据库复制，既可以通过基于用户管理的方式来完成，也可以基于 RMAN 方式来实现。建议使用 RMAN 方式来实现，因为它简单易用，隐藏了复杂的逻辑，仅仅是执行一条 duplicate 命令就可以了。当然，前期的准备工作也是不可缺少的，如创建相应的 dump 目录、准备参数文件、配置监听等。本文描述了 Oracle 11g 下如何使用 RMAN 实现同机克隆数据库。

首选介绍 RMAN 克隆的几种类型。

- 利用 RMAN 备份克隆并访问目标数据库（也就是原数据库），也就是复制期间由 Oracle net 与目标数据库保持连接。
- 利用 RMAN 备份克隆不访问目标数据库，比如网络不通阿，目标数据库不可用等，总之是人为或故障使得与目标库失去连接。
- 直接使用活动数据库（active）进行克隆，实时备份加克隆。

RMAN 克隆做了什么？RMAN 克隆根据需要连接或不连接到目标数据库后，需要连接一个辅助实例。这个辅助实例也就是复制后的实例。知道任何一个数据库至少有一个实例与之对应，如果是 RAC 环境则可以多个实例对应一个数据库。因此，在克隆数据库之前先建一个 NOMOUNT 状态的辅助实例用于分配内存等一系列的后台进程。有了实例之后，RMAN 为这个辅助实例生成控制文件，基于实例来还原数据库、恢复数据库等操作。这些操作依靠 duplicate target Databaseto aux_db 命令来完成。RMAN 如何连接到辅助实例呢？与连接 target 或 catalog 方式类似，可使用 connect auxiliary name/pwd@tnsstring。通常情况下，对于磁盘备份还原操作，RMAN 会自动创建及分配相应的通道。辅助实例也不例外，当然是自动分配辅助通道，磁带介质就麻烦一点了，需要手工来指定其通道、并行度等。

下面简要描述一下 RMAN 克隆不同阶段都做了什么：

- RMAN 确定备份的属性、位置等，也就是备份存在性。
- RMAN 为辅助实例分配通道及辅助通道的参数设置。
- RMAN 还原数据文件到辅助实例（此时使用了目标数据库控制文件）。
- RMAN 构建辅助实例的控制文件。
- 根据需要还原归档日志并进行相应的介质恢复。
- 重置辅助实例的 dbid，并使用 open resetlog 方式打开数据库，此时会创建相应的联机重做日志文件。

在具体的操作方法上，RMAN 克隆大致分为如下几个步骤。

- 备份目标数据库（根据需要克隆类型而定，异机的话 ftp 一下，此步也可以置于步骤 e 之后、f 之前）。
- 创建相应的 dump 文件夹。
- 配置辅助实例参数文件。

- 生成辅助实例密码文件。
- 配置辅助实例监听。
- 实施数据库克隆（辅助实例启动到 NOMOUNT 状态后）。
- 验证结果。

【示例 6-63】克隆数据库

```
--环境:
--目标数据库: sybo3      /u01/Database/sybo3
--辅助数据库: sybo5      /u01/Database/sybo5

[Oracle@Linux3 Database]$ cat /etc/issue
Enterprise Linux Enterprise Linux Server release 5.5 (Carthage)
Kernel \r on an \m

SQL> select * from v$version where rownum<2;

BANNER
------------------------------------------------------------------------------
--
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - Production


1. 备份目标数据库
--生成后续验证克隆成功的测试数据
SQL> insert into t select 'Jackson', 'Transfer DB by rman' from dual;

SQL> commit;

SQL> select * from t;

NAME       ACTION
---------- --------------------
Robinson   Transfer DB
Jackson    Transfer DB by rman

SQL> alter system archive log current;

[Oracle@Linux3 ~]$ rman target /

Recovery Manager: Release 11.2.0.1.0 - Production on Thu Jul 25 08:39:42 2013

Copyright (c) 1982, 2009, Oracle and/or its affiliates.  All rights reserved.

connected to target Database: SYBO3 (DBID=2347733014)

RMAN> backup Database include current controlfile plus archivelog delete input;

piece
handle=/u01/Database/sybo3/flash_recovery_area/SYBO3/backupset/2013_07_25/o1_m
```

```
f_annnn_TAG20130725T083957_8z0wyy9n_.bkp
tag=TAG20130725T083957 comment=NONE
channel ORA DISK 1: backup set complete, elapsed time: 00:00:01

piece
handle=/u01/Database/sybo3/flash_recovery_area/SYBO3/backupset/2013_07_25/o1_m
f_nnndf_TAG20130725T083959_8z0wz06c_.bkp
tag=TAG20130725T083959 comment=NONE
channel ORA DISK 1: backup set complete, elapsed time: 00:01:30
Finished backup at 2013/07/25 08:41:29

Starting Control File and SPFILE Autobackup at 2013/07/25 08:41:31
piece
handle=/u01/Database/sybo3/flash recovery area/SYBO3/autobackup/2013 07 25/o1
mf s 821695291 8z0x1vsf .bkp comment=NONE
Finished Control File and SPFILE Autobackup at 2013/07/25 08:41:34


2. 创建相应的 dump 文件夹
[Oracle@Linux3 Database]$ more sybo5.sh
#!/bin/sh
mkdir -p /u01/Database
mkdir -p /u01/Database/sybo5/adump
mkdir -p /u01/Database/sybo5/controlf
mkdir -p /u01/Database/sybo5/flash_recovery_area
mkdir -p /u01/Database/sybo5/oradata
mkdir -p /u01/Database/sybo5/redo
mkdir -p /u01/Database/sybo5/dpdump
mkdir -p /u01/Database/sybo5/pfile
[Oracle@Linux3 Database]$ ./sybo5.sh


3. 配置辅助实例参数文件
--在 sqlplus 下生成辅助实例的参数文件
SQL> create pfile='/u01/Oracle/db_1/dbs/initsybo5.ora' from spfile;

--修改辅助实例参数文件
$ sed -i 's/sybo3/sybo5/g' $ORACLE HOME/dbs/initsybo5.ora
$ grep sybo3 $ORACLE HOME/dbs/initsybo5.ora    -->校验是否还存在 sybo3 相关字符

--下面是修改后最终的结果
[Oracle@Linux3 Database]$ more $ORACLE_HOME/dbs/initsybo5.ora
sybo5.__db_cache_size=113246208
sybo5.  java pool size=4194304
sybo5.  large pool size=4194304
sybo5.__Oracle_base='/u01/Oracle'#ORACLE_BASE set from environment
sybo5.__pga_aggregate_target=142606336
sybo5.__sga_target=234881024
sybo5.__shared_io_pool_size=0
sybo5.  shared pool size=104857600
sybo5.__streams_pool_size=0
```

```
*.audit_file_dest='/u01/Database/sybo5/adump/'
*.audit_trail='db'
*.compatible='11.2.0.0.0'
*.control files='/u01/Database/sybo5/controlf/control01.ctl',
'/u01/Database/sybo5/controlf/control02.ctl'
*.db_block_size=8192
*.db_domain='orasrv.com'
*.db name='sybo5'
*.db recovery file dest='/u01/Database/sybo5/flash recovery area/'
*.db_recovery_file_dest_size=4039114752
*.dg_broker_config_file1='/u01/Database/sybo5/db_broker/dr1sybo5.dat'
*.dg_broker_config_file2='/u01/Database/sybo5/db_broker/dr2sybo5.dat'
*.dg_broker_start=FALSE
*.diagnostic dest='/u01/Database/sybo5'
*.log archive dest 1=''         #此处未指定 archive 位置，使用默认的闪回区
*.memory_target=374341632
*.open_cursors=300
*.processes=150
*.remote_login_passwordfile='EXCLUSIVE'
*.undo tablespace='UNDOTBS1'


4. 生成辅助实例密码文件
--直接使用 orapwd 命令完成
$ orapwd file=$ORACLE_HOME/dbs/orapwsybo5 password=Oracle entries=10


5. 配置辅助实例监听
--配置辅助实例的监听方式很多，如 netca、netmgr、直接命令方式等，下面直接给出脚本
[Oracle@Linux3 ~]$ more $ORACLE_HOME/network/admin/listener.ora
# listener.ora Network Configuration File:
/u01/Oracle/db 1/network/admin/listener.ora
# Generated by Oracle configuration tools.

SID_LIST_LISTENER_SYBO5 =
  (SID_LIST =
    (SID_DESC =
      (GLOBAL DBNAME = sybo5.orasrv.com)
      (ORACLE HOME = /u01/Oracle/db 1)
      (SID_NAME = sybo5)
    )
  )

SID LIST LISTENER SYBO3 =
  (SID LIST =
    (SID_DESC =
      (GLOBAL_DBNAME = sybo3.orasrv.com)
      (ORACLE_HOME = /u01/Oracle/db_1)
      (SID_NAME = sybo3)
    )
  )
```

```
LISTENER_SYBO5 =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = Linux3.orasrv.com)(PORT = 1532))
  )

ADR_BASE_LISTENER_SYBO5 = /u01/Oracle

LISTENER_SYBO3 =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = Linux3.orasrv.com)(PORT = 1531))
  )

ADR_BASE_LISTENER_SYBO3 = /u01/Oracle

[Oracle@Linux3 ~]$ more $ORACLE_HOME/network/admin/tnsnames.ora
# tnsnames.ora Network Configuration File:
/u01/Oracle/db_1/network/admin/tnsnames.ora
# Generated by Oracle configuration tools.

SYBO5 =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.7.25)(PORT = 1532))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = SYBO5.ORASRV.COM)
    )
  )

SYBO3 =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.7.25)(PORT = 1531))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = SYBO3.ORASRV.COM)
    )
  )


6. 实施数据库克隆
--下面先启动辅助实例到 nomount 状态
[Oracle@Linux3 ~]$ export ORACLE_SID=sybo5
[Oracle@Linux3 ~]$ sqlplus / as sysdba
SQL> startup nomount pfile=/u01/Oracle/db_1/dbs/initsybo5.ora;
ORACLE instance started.

--调用 RMAN 连接到目标数据库与辅助数据库
[Oracle@Linux3 ~]$ rman target sys/Oracle@sybo3 auxiliary sys/Oracle@sybo5
```

```
Recovery Manager: Release 11.2.0.1.0 - Production on Thu Jul 25 14:32:51 2013

Copyright (c) 1982, 2009, Oracle and/or its affiliates. All rights reserved.

connected to target Database: SYBO3 (DBID=2347733014)
connected to auxiliary Database: SYBO5 (not mounted)

RMAN> run{
set newname for datafile 1  to '/u01/Database/sybo5/oradata/system01.dbf';
set newname for datafile 2  to '/u01/Database/sybo5/oradata/sysaux01.dbf';
set newname for datafile 3  to '/u01/Database/sybo5/oradata/undotbs01.dbf';
set newname for datafile 4  to '/u01/Database/sybo5/oradata/users01.dbf';
set newname for datafile 5  to '/u01/Database/sybo5/oradata/example01.dbf';
set newname for tempfile 1  to '/u01/Database/sybo5/oradata/temp01.dbf';
duplicate target Database to sybo5
logfile
group 1 ('/u01/Database/sybo5/redo/redo01a.log',
'/u01/Database/sybo5/redo/redo01b.log') size 10m,
group 2 ('/u01/Database/sybo5/redo/redo02a.log',
'/u01/Database/sybo5/redo/redo02b.log') size 10m,
group 3 ('/u01/Database/sybo5/redo/redo03a.log',
'/u01/Database/sybo5/redo/redo03b.log') size 10m;
switch datafile all;
}

executing command: SET NEWNAME

executing command: SET NEWNAME

executing command: SET NEWNAME

executing command: SET NEWNAME

executing command: SET NEWNAME

executing command: SET NEWNAME

Starting Duplicate Db at 2013/07/25 14:33:14
using target Database control file instead of recovery catalog
---->使用了原数据库的控制文件
allocated channel: ORA_AUX_DISK_1
channel ORA_AUX_DISK_1: SID=134 device type=DISK

contents of Memory Script:         ------->Oracle 会自动生成 auxiliary db 的 spfile
{
   sql clone "create spfile from memory";
}
executing Memory Script

sql statement: create spfile from memory
```

```
contents of Memory Script:   ------>下面的 SQL 使用 spfile 重启 auxiliary db 到 nomount 状态
{
   shutdown clone immediate;
   startup clone nomount;
}
executing Memory Script

Oracle instance shut down

connected to auxiliary Database (not started)
Oracle instance started

Total System Global Area     376635392 bytes

Fixed Size                    1336652 bytes
Variable Size               260049588 bytes
Database Buffers            109051904 bytes
Redo Buffers                  6197248 bytes

contents of Memory Script:          ----->下面的这段 SQL 完成了一系列任务，见后面的分解
{
   sql clone "alter system set  db_name =
 ''SYBO3'' comment=
 ''Modified by RMAN duplicate'' scope=spfile";
   sql clone "alter system set  db_unique_name =
 ''SYBO5'' comment=
 ''Modified by RMAN duplicate'' scope=spfile";
   shutdown clone immediate;
   startup clone force nomount
   restore clone primary controlfile;
   alter clone Database mount;
}
executing Memory Script

sql statement: alter system set  db_name =  ''SYBO3'' comment= ''Modified by RMAN
duplicate'' scope=spfile -->修改 db_name

sql statement: alter system set  db unique name = ''SYBO5'' comment= ''Modified by
RMAN duplicate'' scope=spfile -->修改 db unique name

Oracle instance shut down

Oracle instance started

Total System Global Area     376635392 bytes

Fixed Size                    1336652 bytes
Variable Size               260049588 bytes
Database Buffers            109051904 bytes
Redo Buffers                  6197248 bytes
```

```
Starting restore at 2013/07/25 14:33:29        --->开始 restore
allocated channel: ORA_AUX_DISK_1
channel ORA AUX DISK 1: SID=134 device type=DISK

channel ORA_AUX_DISK_1: starting datafile backup set restore
channel ORA_AUX_DISK_1: restoring control file    --->首先 restore controlfile
channel ORA_AUX_DISK_1: reading from backup piece
/u01/Database/sybo3/flash recovery area/SYBO3/autobackup/2013 07 25/
   o1 mf s 821695291 8z0x1vsf .bkp
channel ORA_AUX_DISK_1: piece
handle=/u01/Database/sybo3/flash_recovery_area/SYBO3/autobackup/2013_07_25/
   o1_mf_s_821695291_8z0x1vsf_.bkp tag=TAG20130725T084131
channel ORA_AUX_DISK_1: restored backup piece 1
channel ORA AUX DISK 1: restore complete, elapsed time: 00:00:01
-->controlfile restore 完成
output file name=/u01/Database/sybo5/controlf/control01.ctl
-->output 到的位置
output file name=/u01/Database/sybo5/controlf/control02.ctl
-->注意此时的控制文件中记录的信息依旧是 sybo3 的，即 target db
Finished restore at 2013/07/25 14:33:30

Database mounted       -->数据库切换到 mount 状态，对应语句 alter clone Database mount

contents of Memory Script:         -->这些 SQL 语句用于设置 auxiliary db 数据文件位置
{
  set until scn  886687;           -->Oracle 自动设定了相应的 scn
  set newname for datafile  1 to
 "/u01/Database/sybo5/oradata/system01.dbf";
  set newname for datafile  2 to
 "/u01/Database/sybo5/oradata/sysaux01.dbf";
  set newname for datafile  3 to
 "/u01/Database/sybo5/oradata/undotbs01.dbf";
  set newname for datafile  4 to
 "/u01/Database/sybo5/oradata/users01.dbf";
  set newname for datafile  5 to
 "/u01/Database/sybo5/oradata/example01.dbf";
  restore                                                 --> 发布 restore 命令
  clone Database
  ;
}
executing Memory Script

executing command: SET until clause

executing command: SET NEWNAME

executing command: SET NEWNAME

executing command: SET NEWNAME

executing command: SET NEWNAME
```

```
executing command: SET NEWNAME

Starting restore at 2013/07/25 14:33:35    -->开始数据文件的 restore
using channel ORA_AUX_DISK_1    -->根据控制文件的信息读取备份集并还原到 set newname 位置

channel ORA_AUX_DISK_1: starting datafile backup set restore
channel ORA AUX DISK 1: specifying datafile(s) to restore from backup set
channel ORA AUX DISK 1: restoring datafile 00001 to
/u01/Database/sybo5/oradata/system01.dbf
channel ORA_AUX_DISK_1: restoring datafile 00002 to
/u01/Database/sybo5/oradata/sysaux01.dbf
channel ORA_AUX_DISK_1: restoring datafile 00003 to
/u01/Database/sybo5/oradata/undotbs01.dbf
channel ORA AUX DISK 1: restoring datafile 00004 to
/u01/Database/sybo5/oradata/users01.dbf
channel ORA_AUX_DISK_1: restoring datafile 00005 to
/u01/Database/sybo5/oradata/example01.dbf
channel ORA_AUX_DISK_1: reading from backup piece
/u01/Database/sybo3/flash recovery area/SYBO3/backupset/2013 07 25/
  o1 mf nnndf TAG20130725T083959 8z0wz06c .bkp
channel ORA_AUX_DISK_1: piece
handle=/u01/Database/sybo3/flash_recovery_area/SYBO3/backupset/2013_07_25/
  o1_mf_nnndf_TAG20130725T083959_8z0wz06c_.bkp tag=TAG20130725T083959
channel ORA_AUX_DISK_1: restored backup piece 1
channel ORA AUX DISK 1: restore complete, elapsed time: 00:01:15
-->完成数据文件 restore
Finished restore at 2013/07/25 14:34:50

contents of Memory Script:           -->下面的脚本将新的数据文件全部更新到控制文件
{
   switch clone datafile all;
}
executing Memory Script

datafile 1 switched to datafile copy
input datafile copy RECID=7 STAMP=821716490 file
name=/u01/Database/sybo5/oradata/system01.dbf
datafile 2 switched to datafile copy
input datafile copy RECID=8 STAMP=821716491 file
name=/u01/Database/sybo5/oradata/sysaux01.dbf
datafile 3 switched to datafile copy
input datafile copy RECID=9 STAMP=821716491 file
name=/u01/Database/sybo5/oradata/undotbs01.dbf
datafile 4 switched to datafile copy
input datafile copy RECID=10 STAMP=821716491 file
name=/u01/Database/sybo5/oradata/users01.dbf
datafile 5 switched to datafile copy
input datafile copy RECID=11 STAMP=821716491 file
name=/u01/Database/sybo5/oradata/example01.dbf
```

```
contents of Memory Script:        -->下面的脚本 Oracle 自动设置了 scn 后发布 recover 命令
{
   set until scn  886687;
   recover
   clone Database
    delete archivelog
   ;
}
executing Memory Script

executing command: SET until clause

Starting recover at 2013/07/25 14:34:51    -->下面使用 archivelog 进行 recover
using channel ORA AUX DISK 1

starting media recovery

archived log for thread 1 with sequence 16 is already on disk as file
/u01/Database/sybo3/flash_recovery_area/SYBO3/
  archivelog/2013 07 25/o1 mf 1 16 8z16rk6o .arc
-->此时运用到了一个 sybo3 已经存在的归档日志，sequence 为 16
channel ORA_AUX_DISK_1: starting archived log restore to default destination
-->接下来还原归档日志到默认位置
channel ORA_AUX_DISK_1: restoring archived log
archived log thread=1 sequence=15
-->从备份的归档日志中读取 sequence 为 15 的
channel ORA AUX DISK 1: reading from backup piece
/u01/Database/sybo3/flash_recovery_area/SYBO3/backupset/2013_07_25/
  o1_mf_annnn_TAG20130725T084129_8z0x1syh_.bkp
channel ORA_AUX_DISK_1: piece
handle=/u01/Database/sybo3/flash_recovery_area/SYBO3/backupset/2013_07_25/
  o1 mf annnn TAG20130725T084129 8z0x1syh .bkp tag=TAG20130725T084129
channel ORA AUX DISK 1: restored backup piece 1
channel ORA_AUX_DISK_1: restore complete, elapsed time: 00:00:01
archived log file
name=/u01/Database/sybo5/flash_recovery_area/SYBO5/archivelog/2013_07_25/o1_mf
_1_15_8z1krh5x_.arc thread=1 sequence=15
channel clone default: deleting archived log(s)      -->删除归档日志
archived log file
name=/u01/Database/sybo5/flash_recovery_area/SYBO5/archivelog/2013_07_25/o1_mf
_1_15_8z1krh5x_.arc RECID=12 STAMP=821716495
archived log file
name=/u01/Database/sybo3/flash_recovery_area/SYBO3/archivelog/2013_07_25/o1_mf
 1 16 8z16rk6o .arc thread=1 sequence=16
media recovery complete, elapsed time: 00:00:04      -->介质恢复完成
Finished recover at 2013/07/25 14:35:00

contents of Memory Script:                    -->下面的脚本用于还原恢复之后的后续工作
{                                             -->包括重新设置 db_name，db_unique_name
   shutdown clone immediate;
   startup clone nomount;
```

```
   sql clone "alter system set  db_name =   ''SYBO5'' comment= ''Reset to original
value by RMAN'' scope=spfile";
   sql clone "alter system reset  db unique name scope=spfile";
   shutdown clone immediate;
   startup clone nomount;
}
executing Memory Script

Database dismounted
Oracle instance shut down

--Author: Robinson
--Blog  : http://blog.csdn.net/robinson_0612

connected to auxiliary Database (not started)
Oracle instance started

Total System Global Area    376635392 bytes

Fixed Size                   1336652 bytes
Variable Size              260049588 bytes
Database Buffers           109051904 bytes
Redo Buffers                 6197248 bytes

sql statement: alter system set  db_name =  ''SYBO5'' comment= ''Reset to original
value by RMAN'' scope=spfile

sql statement: alter system reset  db_unique_name scope=spfile

Oracle instance shut down

connected to auxiliary Database (not started)
Oracle instance started

Total System Global Area    376635392 bytes

Fixed Size                   1336652 bytes
Variable Size              260049588 bytes
Database Buffers           109051904 bytes
Redo Buffers                 6197248 bytes
sql statement: CREATE CONTROLFILE REUSE SET DATABASE "SYBO5" RESETLOGS ARCHIVELOG
-->注意这里，重新创建控制文件
  MAXLOGFILES     16
  MAXLOGMEMBERS      3
  MAXDATAFILES      100
  MAXINSTANCES     8
  MAXLOGHISTORY      292
 LOGFILE
  GROUP  1 ( '/u01/Database/sybo5/redo/redo01a.log',
'/u01/Database/sybo5/redo/redo01b.log' ) SIZE 10 M ,
```

```
  GROUP  2 ( '/u01/Database/sybo5/redo/redo02a.log',
'/u01/Database/sybo5/redo/redo02b.log' ) SIZE 10 M ,
  GROUP  3 ( '/u01/Database/sybo5/redo/redo03a.log',
'/u01/Database/sybo5/redo/redo03b.log' ) SIZE 10 M
 DATAFILE
  '/u01/Database/sybo5/oradata/system01.dbf'
 CHARACTER SET AL32UTF8

contents of Memory Script:
{
   set newname for tempfile  1 to
 "/u01/Database/sybo5/oradata/temp01.dbf";
   switch clone tempfile all;
   catalog clone datafilecopy  "/u01/Database/sybo5/oradata/sysaux01.dbf",
 "/u01/Database/sybo5/oradata/undotbs01.dbf",
 "/u01/Database/sybo5/oradata/users01.dbf",
 "/u01/Database/sybo5/oradata/example01.dbf";
   switch clone datafile all;
}
executing Memory Script

executing command: SET NEWNAME

renamed tempfile 1 to /u01/Database/sybo5/oradata/temp01.dbf in control file

cataloged datafile copy
datafile copy file name=/u01/Database/sybo5/oradata/sysaux01.dbf RECID=1
STAMP=821716521
cataloged datafile copy
datafile copy file name=/u01/Database/sybo5/oradata/undotbs01.dbf RECID=2
STAMP=821716521
cataloged datafile copy
datafile copy file name=/u01/Database/sybo5/oradata/users01.dbf RECID=3
STAMP=821716521
cataloged datafile copy
datafile copy file name=/u01/Database/sybo5/oradata/example01.dbf RECID=4
STAMP=821716521

datafile 2 switched to datafile copy
input datafile copy RECID=1 STAMP=821716521 file
name=/u01/Database/sybo5/oradata/sysaux01.dbf
datafile 3 switched to datafile copy
input datafile copy RECID=2 STAMP=821716521 file
name=/u01/Database/sybo5/oradata/undotbs01.dbf
datafile 4 switched to datafile copy
input datafile copy RECID=3 STAMP=821716521 file
name=/u01/Database/sybo5/oradata/users01.dbf
datafile 5 switched to datafile copy
input datafile copy RECID=4 STAMP=821716521 file
name=/u01/Database/sybo5/oradata/example01.dbf
```

```
contents of Memory Script:
{
   Alter clone Database open resetlogs;
}
executing Memory Script

Database opened
Finished Duplicate Db at 2013/07/25 14:35:36

--验证克隆的结果
[Oracle@Linux3 dbs]$ export ORACLE_SID=sybo5
[Oracle@Linux3 dbs]$ sqlplus / as sysdba

SQL*Plus: Release 11.2.0.1.0 Production on Thu Jul 25 14:38:21 2013

Copyright (c) 1982, 2009, Oracle. All rights reserved.

Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> select * from t;

NAME       ACTION
---------- --------------------
Robinson   Transfer DB
Jackson    Transfer DB by rman

SQL> select name, dbid, open_mode from v$Database;   -->使用了与之前数据库不同的 dbid

NAME           DBID OPEN_MODE
--------- ---------- --------------------
SYBO5     2292457546 READ WRITE
```

6.3 小结

通过本章节的介绍，读者可以了解备份与恢复的具体操作方法。其实 RMAN 工具的使用范围是非常广泛的。RMAN 备份恢复工具是数据库管理员需要重点掌握并且需要熟练使用的工具，不仅仅用在备份和恢复的场景，在数据库迁移、数据库复制等方面也发挥着重要的作用，希望读者可以加强这方面的练习。

正常情况下，数据库的备份恢复工作不会占用 DBA 太多时间。在日常工作中，大部分的时间是需要处理各种数据库故障或者性能问题，性能优化的工作是衡量一位 DBA 是否对 Oracle 具备深入了解的方法，关于 Oracle 数据库的性能管理，将在下一章节向读者进行阐述。

第 7 章

管理数据库性能

数据库是信息系统中核心的部分，数据库的高效性、安全性、稳定性、延展性是项目成功的关键之一。一个好的数据库系统，设计当然是关键，但是就像显微镜的粗调和微调，当软件开发完成后，通过不断地对系统的跟踪监控，对数据库系统进行优化也是必不可少的。可以认为优化是数据库设计的一种顺延，和设计是相辅相成的。数据库的优化实际上从设计、数据库的安装就已经开始了，不应该就 SQL 语句的效率孤立地谈论数据库的优化问题。实际上很多数据库优化的经验在设计时就必须要考虑了。

大体上，Oracle 数据库性能管理中的三项活动可分为：

- 性能规划
- 实例优化
- SQL 优化

性能规划是建立环境的过程，包括硬件、软件、操作系统、网络基础结构等。规划时必须权衡性能（速度）、开销和可靠性。必须考虑在系统体系结构方面的投资：满足需求所需的硬件和软件基础结构。当然，这需要通过分析来确定适合于给定环境、应用程序和性能需求的值。例如，硬盘驱动器和控制器的数量会影响数据的访问速度。应用程序的伸缩能力也是非常重要的。这意味着能够处理越来越多的用户、客户机、会话或事务处理，而不会对系统整体性能产生大的影响。

限制可伸缩性的最明显因素是用户间的串行化操作。如果所有的用户同时都经过同一路径，则在添加更多用户时，毫无疑问地会对性能造成负面影响。这是因为会有越来越多的用户排队等待通过该路径。编写的 SQL 语句质量差也会影响可伸缩性。这会使许多用户都在等待低效 SQL 的完成，每个用户都在与其他用户竞争大量资源，但实际上他们并不需要这些资源。

应用程序的设计原理会对性能产生重大影响。设计的简洁性、视图和索引的使用以及数据建模都是非常重要的。任何应用程序都必须在典型的生产工作量下进行测试。这需要对数据库大小和工作量进行估计，并生成测试数据和系统负载。部署新的应用程序（或应用程序的新版本）时必须考虑到性能。有时，需要在推广新应用程序的过程中制定设计决策，使其与旧系统保持兼容。新的数据库应该根据生产环境，为其承载的应用程序专门进行配置。

一项困难但必须完成的任务是在更改基础结构时测试现有的应用程序。例如，将数据库升级到更高版本，或者更换操作系统或服务器硬件。以新的配置将应用程序部署到生产中以前，需要了解这样做的影响。一般几乎都需要对应用程序进行额外的优化。需要确定关键的功能会

正常运行，不会发生错误。

实例优化是对 Oracle 数据库参数和操作系统（OS）参数进行实际调整，从而使 Oracle 数据库获得更好的性能。任何优化活动在开始时都必须有具体的目标。例如，“每分钟处理 500 个销售事务”，这样的目标要比“使处理速度尽可能快，然后就可以知道要达到什么程度”之类的目标更容易实现。必须为应用程序分配合适的 Oracle 数据库内存，以获得最佳性能。可以使用的内存是有限的。为 Oracle 数据库的某些部分分配的内存太少会导致后台操作的效率很低。如果不进行分析，甚至可能都意识不到这个问题。磁盘 I/O 通常会成为数据库的瓶颈，因此任何数据库实施的开始阶段都需要特别注意。操作系统的配置也会影响 Oracle 数据库的性能。

SQL 优化涉及的工作是使应用程序提交有效的 SQL 语句。SQL 优化可针对整个应用程序执行，也可针对单条语句执行。在应用程序级别，需要确保应用程序的各个部分都能够利用彼此的工作，并且不会产生不必要的资源竞争。

7.1 几个与性能管理相关的概念

7.1.1 性能优化数据

Oracle 数据库服务器软件可以获取与自身运行有关的信息。收集的数据主要有三种类型，即累计统计信息、度量和抽样统计信息。累计统计信息是数据库服务器中发生的各种事件的计数信息和计时信息，有些信息非常重要，如缓冲区忙等待；有些信息对优化几乎没有影响，例如索引块拆分。对于优化而言，最重要的事件通常是显示的累计时间值最大的事件。Oracle Database 11g 中的统计信息通过使用时间模型进行关联。时间模型统计信息基于数据库时间的百分比，为进行比较提供了一个共同的基础。度量是每单位的统计计数，单位可能是时间（如秒）、事务处理或会话。度量为主动监视性能提供一个基础，并可以为导致生成预警的度量设置阈值，例如将阈值设置为每毫秒的读取次数超过以前记录的峰值时或归档日志区已占用 95% 时。如果将 STATISTICS_LEVEL 设置为 TYPICAL 或 ALL，就会自动收集抽样统计信息。利用抽样统计信息可以及时进行回顾，也可以在不同的维查看过去收集的会话和系统统计信息，即使事先未考虑过指定对这些维收集数据。

7.1.2 优化统计信息收集

优化程序统计信息是有关数据库对象的特定详细资料的数据集合。查询优化程序必须使用这些统计信息为每个 SQL 语句选择最佳的执行计划，还会定期收集这些统计信息，在两次收集间隔之内数据不会发生变化。建议允许 Oracle 数据库自动搜集统计信息，以此来搜集优化程序统计信息。自动维护任务是在创建数据库时自动创建的，并由调度程序进行管理。默认情况下，它会搜集数据库中优化程序统计信息缺失或已过时的所有对象的统计信息。可以通过

“Automatic Maintenance Tasks（自动维护任务）”页更改默认配置。

系统统计信息为查询优化程序描述系统的硬件特征，例如 I/O 及 CPU 性能和利用率。选择执行计划时，优化程序会估计每个查询所需的 I/O 和 CPU 资源。通过系统统计信息，查询优化程序可以更加准确地估计 I/O 和 CPU 开销，从而选择更好的执行计划。系统统计信息是使用 DBMS_STATS.GATHER_SYSTEM_STATS 过程收集的。搜集系统统计信息时，Oracle 数据库对指定时段内的系统活动进行分析。系统统计信息不是自动搜集的。Oracle 建议使用 DBMS_STATS 程序包来搜集系统统计信息。

如果选择不使用统计信息自动搜集功能，就必须手动收集所有方案中的统计信息，包括系统方案。如果数据库中的数据定期发生更改，就需要定期搜集统计信息，以确保统计信息能够准确地反映数据库对象的特征。要手动收集统计信息，请使用 DBMS_STATS 程序包。此 PL/SQL 程序包可以用于修改、查看、导出、导入和删除统计信息。还可以通过数据库初始化参数来管理优化程序和系统统计信息收集，例如：

- OPTIMIZER_DYNAMIC_SAMPLING 参数控制优化程序执行的动态采样级别。当表和相关索引的统计信息不可用或因为过于陈旧而不可信时，可以使用动态采样来进行估计。当收集到的统计信息无法使用或可能会导致重大估计错误时，动态采样也可以估计单表的谓词选择性。
- STATISTICS_LEVEL 参数控制数据库中所有主要统计信息的收集或指导，并设置数据库的统计信息收集级别。此参数的值包括 BASIC、TYPICAL 和 ALL。通过查询 V$STATISTICS_LEVEL 视图可以确定受 STATISTICAL_LEVEL 参数影响的参数。



将 STATISTICS_LEVEL 设置为 BASIC 时将禁用许多自动功能，因此不建议使用该项。

7.1.3　Oracle 等待事件

等待事件是按服务器进程或线程递增的统计信息，指示必须等待事件完成之后才能继续处理。等待事件数据显示了可能会影响性能的问题症状，如闩锁争用、缓冲区争用和 I/O 争用。记住，这些只是问题的症状，而不是实际的原因。等待事件按类别进行分组，包括管理、应用程序、簇、提交、并发、配置、空闲、网络、其他、调度程序、系统 I/O 和用户 I/O。

Oracle 数据库中有 800 多种等待事件，包括“free buffer wait（空闲缓冲区等待）”“latch free（闩锁释放）”“buffer busy waits（缓冲区忙等待）”“db file sequential read（数据库文件顺序读取）”和“db file scattered read（数据库文件分散读取）”。使用 EM，可以通过打开“Performance（性能）”页并查看“Average Active Sessions（平均活动会话数）”来查看等待事件。通过单击特定等待事件类别的链接，可以使用“Top Activity（顶级活动）”界面追溯至特定的等待事件。在本例中，有一个很小的“buffer busy waits（缓冲区忙等待）”集。

7.1.4 实例统计信息

要有效地诊断性能问题，必须有统计信息可供使用。Oracle 数据库实例为系统、会话和单条 SQL 语句在实例级别生成许多类型的累计统计信息。Oracle 数据库还跟踪段和服务的累计统计信息。分析其中任一范围内的性能问题时，通常需要查看所关注时段内统计信息的变化（差值）。



实例统计信息是动态的，在每个实例启动时进行重置。可以在某个时间点捕获这些统计信息，并将其以快照形式存储在数据库中。

- 等待事件统计信息

所有可能存在的等待事件均列入 V$EVENT_NAME 视图目录。

所有会话的累计统计信息都存储在 V$SYSTEM_EVENT 中，该视图显示特定事件从实例启动以来的等待事件合计。

在排除故障时，需要了解进程是否曾等待某项资源。

- 系统范围的统计信息

所有系统范围的统计信息均列入 V$STATNAME 视图目录，Oracle Database 11g 提供了 400 多种统计信息。

服务器在 V$SYSSTAT 视图中显示所有计算的系统统计信息。可以通过查询此视图，找到自实例启动以来的累计合计。

【示例 7-1】查询视图

```
SQL>  select name ,class ,value from v$sysstat;
NAME CLASS      VALUE
------------------------------- ------ ----------

table scans (short tables)         64     135116
table scans (long tables)          64        250
table scans (rowid ranges)         64          0
table scans (cache partitions)      64          3
table scans (direct read)          64          0
table scan rows gotten            64   14789836
table scan blocks gotten           64     558542
```

系统范围的统计信息按优化主题和调试用途分类。这些类别包括一般实例活动、重做日志缓冲区活动、锁定、数据库缓冲区高速缓存活动等。每条系统统计信息都可能属于多种类别，因此无法对 V$SYSSTATS.CLASS 和 V$SYSTEM_WAIT_CLASS.WAIT_CLASS#进行简单的连接。还可以通过查询 V$SYSTEM_WAIT_CLASS 来查看特定等待类别的所有等待事件，例如：

```
SQL> select * from v$system wait class  where wait class like '%i/o%';
CLASS ID   CLASS# WAIT CLASS   TOTAL WAITS TIME WAITED
---------- ------ ------------ ----------- -----------
1740759767      8 User I/O        1119152      39038
4108307767      9 System I/O       296959      27929
```

关于 SGA 全局统计信息，服务器在 V$SGASTAT 视图中显示所有计算的内存统计信息。

可以通过查询此视图，找到自实例启动以来关于 SGA 使用率的累计合计的详细资料，例如：

```
SQL> select * from v$sgastat;
POOL          NAME                          BYTES
------        -------------------------    ----------
fixed sga 7780360
buffer cache                   25165824
log buffer                      262144
shared pool    sessions                     1284644
shared pool    sql area                    22376876
...
```

上述结果只是输出显示的一部分。若将 STATISTICS_LEVEL 参数设置为 BASIC，则 TIMED_STATISTICS 参数的默认值为 FALSE。这样便不会收集等待事件的计时信息，并且将禁用数据库的许多性能监视功能。显式设置的 TIMED_STATISTICS 值会覆盖从 STATISTICS_LEVEL 派生得到的值。

7.1.5　与会话和服务有关的统计信息

通过查询 V$SESSION 可以查看每个已登录用户的当前会话信息。例如，可以使用 V$SESSION 来确定会话是代表用户会话还是由数据库服务器进程（BACKGROUND）创建的。可以查询 V$SESSION 或 V$SESSION_WAIT 来确定活动会话所等待的资源或事件。可以在 V$SESSTAT 中查看用户会话统计信息。V$SESSION_EVENT 视图列出了会话等待事件的信息。实例统计信息的累计值通常位于动态性能视图中，如 V$SESSTAT 和 V$SYSSTAT。注意，在关闭数据库实例时将重置动态视图中的累计值。V$MYSTAT 视图可显示当前会话的统计信息，还可以通过查询 V$SESSMETRIC 来显示所有活动会话的性能度量值。此视图列出了 CPU 利用率、物理读取数、硬分析数和逻辑读取比率等性能度量。

在应用程序服务器共享数据库连接的 n 层环境中，查看会话可能无法获取分析性能所需的信息。将会话按服务名称分组可以更加准确地监视性能。V$SERVICE_WAIT_CLASS 和 V$SERVICE_EVENT 两个视图提供的信息与名称相似的对应会话视图所提供的信息相同，不同之处在于所提供的是服务层的信息而不是会话层的信息。V$SERVICE_WAIT_CLASS 显示每个服务的等待统计信息，这些信息按等待类别细分。V$SERVICE_EVENT 显示的信息与 V$SERVICE_WAIT_CLASS 的相同，只不过这些信息按照事件 ID 进一步细分。Oracle Enterprise Manager 还按服务以及模块和服务提供信息汇总。可以单击每个视图中的图例，查看每项服务的活动和统计信息。可以使用 DBMS_SERVICE 程序包在数据库中定义服务，并使用网络服务名将应用程序分配到某项服务。

7.2　Oracle 数据库优化方案

Oracle 性能管理既是一种艺术，也是一种科学。从实用角度讲，它可以分为两种类型，主动式和被动式性能管理。主动式性能管理涉及特定系统实施初期的设计和开发，包括硬件选择、

性能及容量规划，海量存储系统的选择，IO 子系统配置及优化，以及如何对不同组件进行定制，以满足 Oracle 数据库和应用系统的复杂要求。被动式性能管理涉及现有环境中不同组件的性能评估、故障排除和 Oracle 环境的优化。

本文旨在探讨如何进行被动式性能调优，以便为 Oracle 性能调优提供必要的指导，从而避免仅仅通过反复尝试的方式进行性能调优，提高 Oracle 性能管理的效率。所以 Oracle 数据库性能恶化表现基本上都是用户响应时间比较长，需要用户长时间的等待。

获得满意的用户响应时间有两个途径：一是减少系统服务时间，即提高数据库的吞吐量；二是减少用户等待时间，即减少用户访问同一数据库资源的冲突率。对于以上两个问题，通常采用以下几个方面来进行改善：调整服务器内存分配。例如，可以根据数据库运行状况调整数据库系统全局区（SGA 区）的数据缓冲区、日志缓冲区和共享池的大小；还可以调整程序全局区（PGA 区）的大小。调整硬盘 I/O 问题，达到 I/O 负载均衡。调整运用程序结构设计优化调整操作系统参数和使用资源管理器 SQL 优化、诊断 latch 竞争、Rollback(undo)Segment 优化、提升 block 的效率等。

7.2.1 内存优化

Oracle 基本的体系结构如图 7-1 所示，每个后台进程将占用 5MB 大小的内存。通过合理地分配内存大小，合理地设置表空间体系和内部空间参数，可以提高磁盘空间的利用率、减少数据段碎片，并且在查询和向数据文件写入数据时使用较少的 IO，较少的 IO 也会降低 CPU 的资源消耗。同时对环境参数的合理配置，可以使数据库中数据顺畅地流动，减少锁存器冲突和各种等待，充分地利用系统资源。

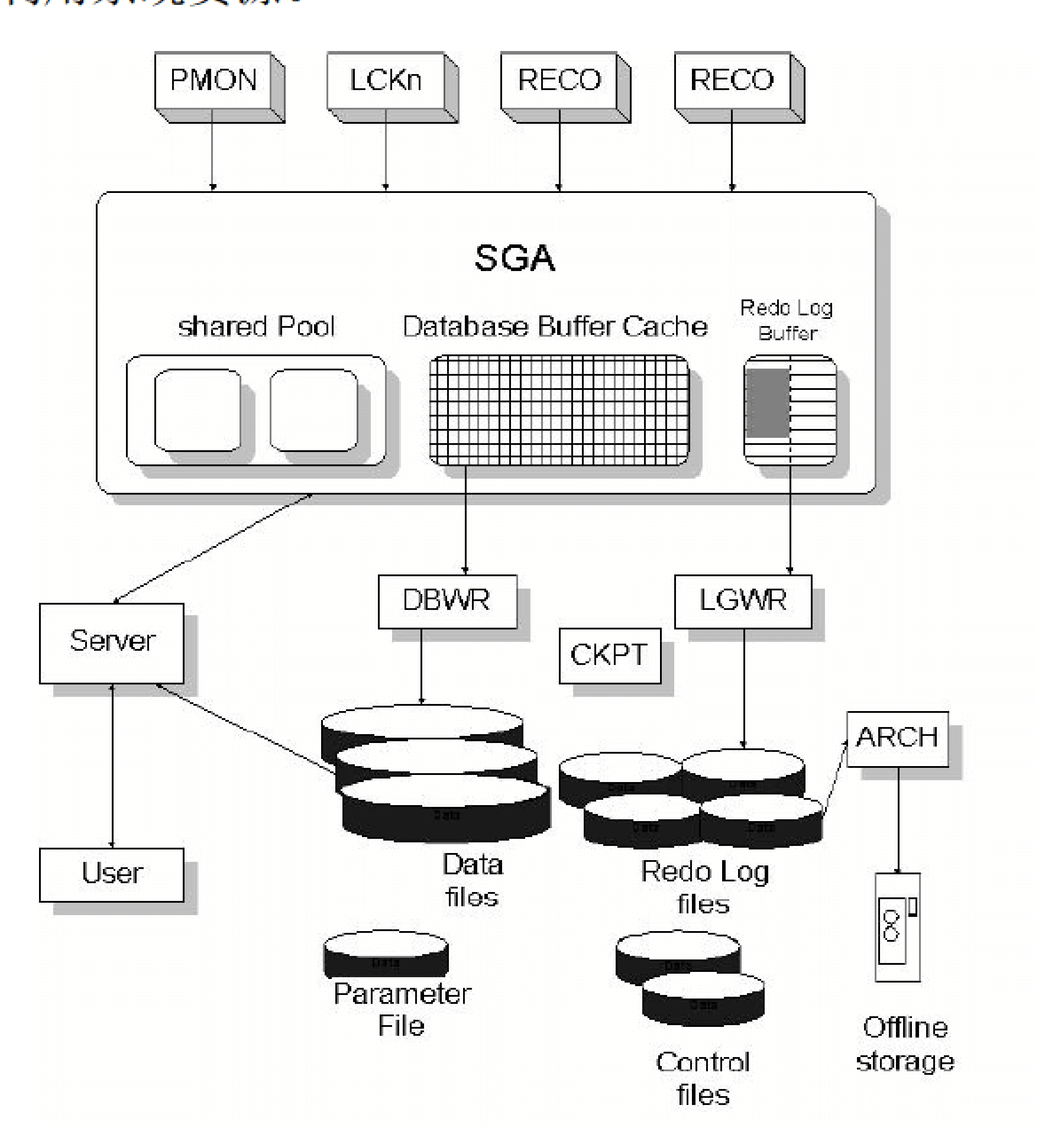


图 7-1 数据库体系结构

内存结构的优化主要是通过 init.ora 文件的环境参数来配置，主要注意下面几个：

- db_block_size：数据库中每个数据块的大小，默认是 2048 字节（2KB），一般应该增大到 4KB、8KB，大型数据库也常使用 16KB 和 32KB，通常 SGA 也应该增加。
- shared_pool_size：至于 shared_pool_size 大小是否合适，可以通过对数据库的监控得到，也可以通过一些 SQL 语句实现。可以查询 v$SGASTAT，查看 SGA 结构，其中 free memory 作为一个估计性的指标，如果大于 20%可以将 shared_pool_size 给小些，给其他部分多分配些，如果小于 8%，就可以将 share_pool_size 开大些。有一些更精确的 SQL 脚本可以查看其中 library_cache、cursor_cache、pl/sql cache 大小是否合适，如果数据库规模增长，还需要多少内存，所有这些都可以在附件中找到。
- log_buffer：为了减少 LGWR 和 DBWR 冲突，大型数据库的 log_buffer 一般都要手动调大些，一般为 2MB 到 3MB。可以查询 V￥SYSSTAT 视图中 redo log space requests，如果大于 0 就应该增加 log_buffer 的值。其他对 Oracle 影响很大的参数有 db_writes、db_file_multiblock_read、sort_direct_writes、sort_area_size。讲起来就多了，由于和例子关系不大所以不再叙述。需要注意的是：对 Oracle 9i 来说，sort_area_size、sort_area_retained_size、hash_area_size、bitmap_merge_area_size 等参数都被放弃，由 pga_aggregate_target 统一动态分配内存区域大小，对于 undo（回滚）操作也有些变化：undo_retention 表示已提交数据在回滚表空间中保留时间，以秒为单位，默认 900。此选项可使新事务尽可能使用空闲的回滚表空间，这样就减少了查询过程因 snapshot too old 而失败的概率。undo_tablespace 表示系统的回滚表空间。所有的环境参数都可以通过系统的监控工具来分析是否适宜。

下面将从内存管理、Row chaining 与 Row migration、实体文件规划、SQL 优化四个角度，探讨如何处理 Oracle 数据库系统效率与管理问题。

假设一种性能问题的场景，某大型公司在过去 10 年业绩增长显著，人员也由 1000 来人增长至 10000 人，7 年前导入以 Oracle 为数据库的 ERP，经由不断的修修补补，功能方面也已发展成熟。不过由于公司持续的扩厂，使用 ERP 的人员不断增加，最近 ERP 的使用反而发生许多不稳定状况，包括：

- 整体运作速度变慢。
- ERP 明明没问题，但就是会报错，特别是人多时。
- 经查 ERP 主机的内存充足，理论上不应该发生这样的问题。
- 相关插件系统很容易被 Oracle 自动断线或是连不上。

排除网络原因或 ERP 有 Bug 因素后，还需再考量的就是数据库问题。但对大部分不熟悉 Oracle 数据库的人员来说，数据库优化成为一项工作难点。下面先简要说明几点 Oracle Database 与其他品牌数据库相比的不同之处：

- Oracle Database Server 中一个数据库就是一个 Instance。
- Oracle Database 使每一个数据库都有各自的内存。

- Oracle 目前需要由 DBA 自行优化。
- Oracle 有很多参数需要进行数据库优化与控制。

针对以上案例，数据库管理员应如何在日常的管理中避免上述问题的发生，首先从内存管理入手。

1. 内存管理

确认操作系统以及 Oracle Database 版本信息，不同的 Oracle Database 版本所提供的优化机制也各不相同，例如 Oracle 9i 提供各自 SGA、PGA 优化参数，但到了 Oracle 12c，则提供了更进阶的整体内存优化参数。不同操作系统以及版本也一样会影响优化方式，例如 Oracle 9i 在 Linux 的使用上，如果为 32 位版本，则 SGA 有 1.7GB 使用的限制，但如果为 64 位版本，则无此限制。在谈内存使用这个问题之前，先要了解 Oracle Database 的内存组成：System Global Area（以下简称 SGA）与 Program Global Area（以下简称 PGA），如图 7-2 所示。

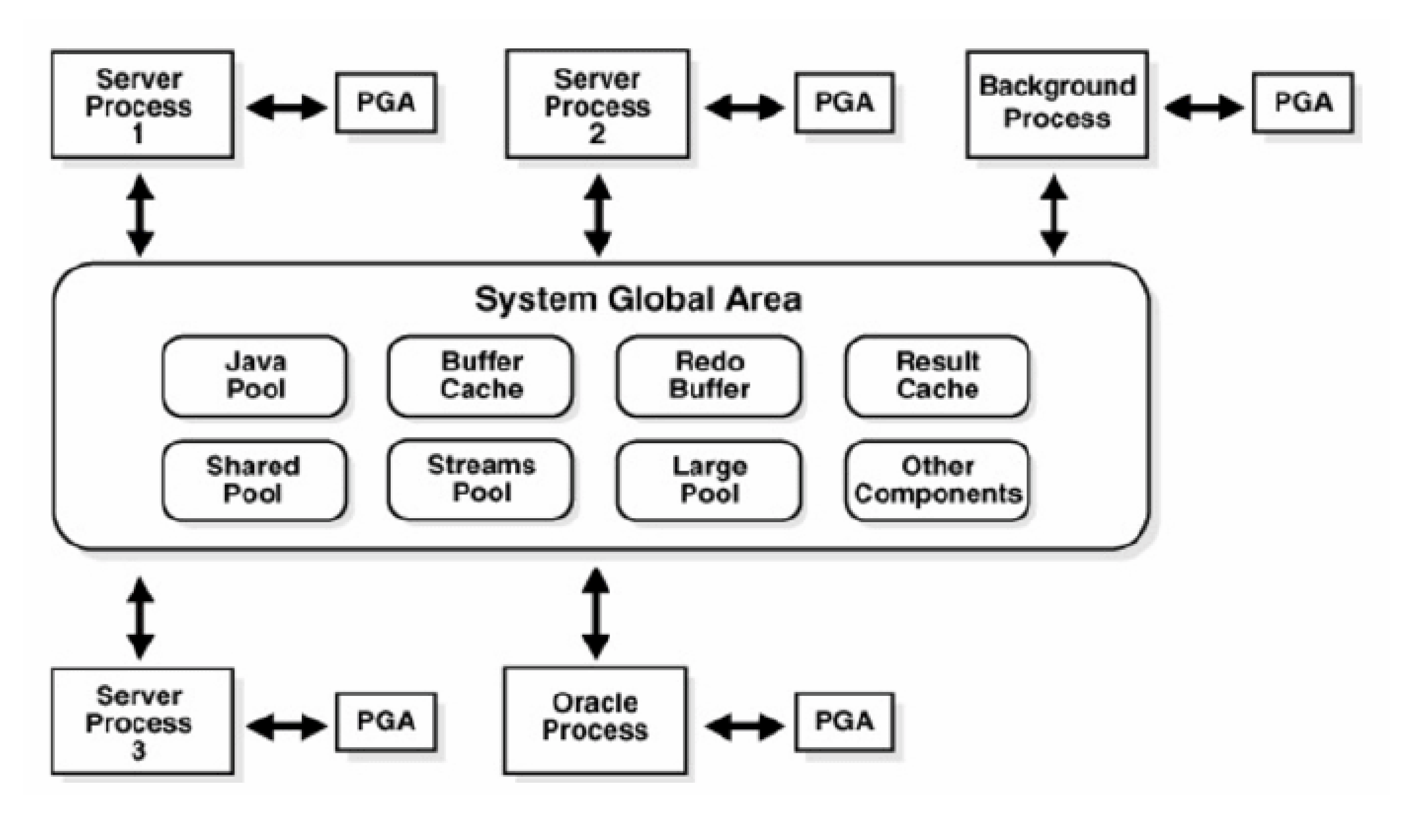


图 7-2　SGA 与 PGA

SGA 中最重要的 3 个组成为：

- database buffer cache：储存从实体文件中读出来的数据。
- redo log buffer：储存用户改变数据的信息，当用户进行 roll back 时即是靠 redo log buffer 的数据。
- shared pool：包含 library cache（储存执行 SQL 过程所需的收集、分解、解析 SQL 所有信息）、dictionary cache（储存有关于 Table、View 等结构的相关信息）、result cache（储存执行结果）。

这些是给 Oracle Instance 共用的内存，所有用户在进行数据库存取时都会共用 SGA。至于 PGA 则如其名，是给每一个 Session 所对应的 background process 或 server Process 使用，每一个 Session 都对应一块私有内存，彼此互不共用，当 Session 结束时 PGA 也会被回收。PGA 是用来执行 Process SQL statement 的，并包含用户登录等信息，主要涉及以下 2 部分：

- session memory：储存有关于 Session variable 的信息，例如用户登录信息。

- private SQL area：包含变量数据、查询状态、查询执行结果存放区（query execution work area，例如 SQL 中的 order by、group by 运行时的数据暂存区）。

从以上的理论说明中可以明确感受到：若 SGA 太小，则与数据读取有关的速度会变慢；PGA 太小，则 SQL 运算特别是与 order by、group by 有关的运算会变慢。那么这与有多少程序会连接到 Oracle 有何关系呢？

程序建立 Session，Session 建立一个或多个 Process（或一个 Process 对应多个 Session），每一个 Process 会耗用一定的 PGA，但在 Oracle 设计中 PGA 的总内存是被参数所控制的，所以 Session 越多、Process 越多，每一个 Process 所能分到的 PGA 就会越少，SQL 的运算就会变慢!Oracle 针对 Process 数量可以进行参数设置，而经由此参数又可以限定所能连接的 Session 数，一般算法为 sessions=1.1*processes + 5。

Process 与 Session 这两者互为一体两面。Process 设置太大，就要给予更多的 PGA，否则程序的 SQL 运算会变慢。如果某天你的用户连线到 Oracle Instance 拒绝连线，或许就是因为 Process 设置不够大，导致可以连线的 Session 不够多。

有关于目前 Oracle Instance 中有多少 Session，可以用以下 SQL 抓取：

```
select count(*) from v$session
```

有关于目前 Oracle Instance 中有多少 Process，可以用以下的 SQL 抓取：

```
select count(*) from v$process
```

该如何配置内存？该配多少内存？Oracle 配置内存的方法依不同版本而有不同做法：例如 Oracle 10g 以后的版本，你可能只要设置总内存参数 memory_target，就可由 Oracle Instance 自行进行内存配置；但在 Oracle 9i 则是 SGA 与 PGA 的内存需各自设置；或是不管在哪一个版本，你都可以进行全手动设置！

当主机有足够内存时，内存配置不会是大问题（只要把所有与内存有关的参数都放大即可），但若主机的总内存不多，或是如前述因版本问题，碰到 SGA 内存不得超过 1.7GB 限制、session 总数破 2000 时，内存的管理可能就是大问题。建议采用 SGA 与 PGA 分开优化的半自动内存管理策略较有弹性，在此情况下不使用 memory_target 参数，而改用以下参数设置：

（1）sga_max_size，设置分配给 SGA 的总内存上限。当 sga_max_size 的内存量大于分配给 SGA 相关参数内存总和时，多出来的部分会分配给 share_pool_size：

- buffer_cacahe_size：理论上越大越好，日后读取相同数据时，可以由内存中读取，不需读取实体文件。
- shared_pool_size：理论上越大越好，日后有相同的 SQL 需要执行时就不需要重新解析。
- log_buffers：对数据库的任何修改都按顺序被记录在该缓冲，然后由 LGWR 进程将它写入 Disk，但正常情况下 LGWR 的写入条件中包含有一条大于 1MB 重做日志缓冲区未被写入 Disk，因此可以说大于 1MB 的 log buffer 值意义并不大。

（2）pga_aggregate_target，设置 PGA 可用总内存数量目标（也就是说，实际 Oracle Instance 使用时有可能超出一点点，但不可以超过太多）。当 pga_agreegate_target 够大时，每一个用户所分配的 private sql area 就会比较大，如有 order by 或 group by 时计算自然较快。

例如 pga_aggregate_target 设为 300MB，5 个用户连接时每个用户可能分发 10MB 的 PGA 内存，共分发 50MB 的 PGA 内存；300 个用户连接时每个用户可能分发 1.3MB 的 PGA 内存，共分发 390MB（当用户连接多时，Oracle 会降低每个用户的 PGA 内存使用量）。

另外，如果你是使用 Oracle 11g 或 12c 等较高版本，并有安装 Oracle Enterprise Manager，则可以引用 Oracle Enterprise Manager 的数据来进行 SGA、PGA 内存的规划（当内存增加并无法降低数据读取次数或提高执行速度时，代表该点即为内存设置的最佳值）。

2. Row chaining（行链接）与 Row migration（行迁移）

关于 Row chaining、Row migration 对数据读写速度的影响，在现实的运作环境中，数据并非如字典一般规矩地排列在实体数据库中。至少有以下因素会导致数据的存放异常：

- Row chaining：当插入一笔较长的数据但该 Data block 又无法容纳该笔数据时，则该笔数据会存放在一个或多个 Data block。日后要读取该数据时，则需读取多个 Data block 造成速度变慢。
- Row migration：当 Data block 将用完时，如果对已有的数据 Update 为较长的数据，则 Oracle 会将原有的数据移到一个新的 Data block，而只在原有的数据空间存放一个指针（但不存放数据）指到新的存放位置。日后要读取该数据时，则需读取多个 Data block（读到原有的 Data block，再经由指针读到真正数据存放处）造成速度变慢。

Row chaining 与 Row migration 会让 Insert、Update 的动作变慢（需跨多个 Data block），也会让 Select 的过程读取更多实体数据，但在实务上又无法避免。一般而言，可以采用以下方法来改善：

- 加大 Data block。例如 Data block 为 4KB，但 Row 的平均长度为 6KB 时，加大 Data block 可以有效地改善 Row chaining 的问题。需注意的是：已建立的 Tablespace，Data block 的大小是不能再更改的，如果要进行这样的处理，你所能做的是：设置参数档 db_nk_cache_size 参数以满足读取较大空间的需求；建立新的 Tablespace，并在建立过程声明较大的 Data block；将这类的 Table 移动、指定存放到该 Tablespace。
- 加大 PCTFREE 参数的大小。将 Table 的 PCTFREE 放大，多保留一些空间供日后 Update 使用，此方法可以降低 Row migration 的情况。
- 针对 Row chaining 或 Migration 严重的 Table 定期进行 Export、Import。有关于 Table row chaining 或 Row migration 的情况，可以利用以下 SQL 抓取数据分析。



对 Oracle Instance 而言，Row migration 被视为 Row chaining 的一种特例，并没有独立分类。

3. 文件规划

除了对上述配置的规划之外，还需要进行实体文件规划，典型的 Oracle 实体文件组成包括如下部分：

- datafile：负责储存来自 User 所输入的数据、相关索引以及其他 DDL、DML 数据，每一个数据库都会有一个或多个 Datafile。当 User 有存取数据需求时会经由 Oracle Instance 先进行需求分析，再由 Oracle Instance 内的 Process 进行 Tablespace 所对应的 Datafile 数据的存取。
- redo log file：一份交易的记录信息，每当 Oracle Instance 对数据库进行数据的新增、删除、修改时，就会将该交易记录写入 Redo log file。若系统发生异常状况，系统即可 redo log file 中的信息，针对 Oracle 失效当时的交易进行 forward commit 或 roll back commit 的 Recovery 处理。
- archived file：已归档、已 Commit 交易过程记录文件（已满的 redo log file）。当有 Recovery 需求时，Oracle 可以参考 Archived file 的交易过程记录信息，将 Oracle 数据库 Recovery 到特定时期。
- control file：储存的是有关 Oracle 的相关核心信息，如有哪些 Datafile、Control file、名称为何、存放于何处。Oracle 于启动过程会读取本档的数据。
- initora_db.ora/spfile：所存放的信息包括建立数据库过程需参考的信息，以及后续数据库的运作信息。
- alter file/trace file：存放预警、追踪的信息。每一个 Server 与 Background process，都可以将相关的警示与错误信息写入 trace file 或 log file。

这些实体文件的存放位置对于数据库的运作速度也会有影响。一般而言，建议将 Datafile 分别建立在不同的硬碟上，例如存放 Index 的 Tablespace 所对应的 Datafile 存放在 A 硬盘，存放数据的 Tablespace 所对应的 Datafile 存放在 B 硬盘，这样在多人连线、多用户存取时由于各硬碟可以同时运作存取数据，因此会有更好的效率。

另外，Datafile、Redo log file、Archived file 最好不要放在同一个硬盘，以避免同一时间、同一硬盘要同时读写不同数据。同时 Redo log file 及 Archived file 都是作为数据库复原的重要依据，如果将这 3 种数据实体文件存放在同一个硬盘，当硬盘损毁时将会造成数据无法复原。

4. SQL 优化

以上均是数据库层面的优化规划思路，在代码层面，主要是针对 SQL 和表这些逻辑层面的优化考虑，首先需要从如何写好 SQL 开始。从 Oracle Instance 的角度来看，“好”SQL 代表能善用 DBA 已建立的 Index、尽可能不要有 Table scan 动作、能有少一点的实体文件读取。另一方面，资深的程序开发人员以及 DBA 通常也都会遵守以下规则：

- Primary key 建立 Index。
- Where 引用到的栏位建立 Index。
- Join 过程中引用的条件建立 Index。

- 必要时以 Cluster 建立 Table。
- 使用多一点的 Where 条件，以让 SQL 少抓一些数据。
- Select 的过程，指名栏位名称，少用 Select *之类的写法。

7.2.2 IO 优化

要正确地的优化 SQL，需要快速定位性能瓶颈点，也就是说快速找到 SQL 主要的开销在哪里。而大多数情况下性能最慢的设备会是瓶颈点，如下载时网络速度可能会是瓶颈点，本地复制文件时硬盘可能会是瓶颈点。为什么这些一般的工作能快速确认瓶颈点呢？因为对这些慢速设备的性能数据有一些基本的认识，如网络带宽是 2Mbps、硬盘是每分钟 7200 转等。因此，为了快速找到 SQL 的性能瓶颈点，也需要了解计算机系统的硬件基本性能指标。

计算机系统硬件性能从高到低依次为 CPU——Cache(L1-L2-L3) ——内存——SSD 硬盘——网络——硬盘。

由于 SSD 硬盘还处于快速发展阶段，因此本文的内容不涉及 SSD 相关应用系统。

根据数据库知识，可以列出每种硬件主要的工作内容：

- CPU 及内存：缓存数据访问、比较、排序、事务检测、SQL 解析、函数或逻辑运算。
- 网络：结果数据传输、SQL 请求、远程数据库访问（dblink）。
- 硬盘：数据访问、数据写入、日志记录、大数据量排序、大表连接。

根据当前计算机硬件的基本性能指标及其在数据库中的主要操作内容，可以整理出如下性能基本优化法则。这个优化法则归纳为 5 个层次：

- 减少数据访问（减少磁盘访问）。
- 返回更少数据（减少网络传输或磁盘访问）。
- 减少交互次数（减少网络传输）。
- 减少服务器 CPU 开销（减少 CPU 及内存开销）。
- 利用更多资源（增加资源）。

在 DBA 日常的工作中，针对 IO 的压力处理是最频繁的，也是最常见的情况。处理 IO 问题比较有效的手段就是在索引上下功夫。

1. 数据库索引

数据库索引的原理非常简单，但在复杂的表中真正能正确使用索引的人很少，即使是专业的 DBA 也不一定能完全做到最优。

为什么不能做到完全的最优？因为索引在提高查询效率的同时也会大大增加表记录的 DML（Insert，Update，Delete）开销，正确的索引可以让性能提升 100 倍以上，不合理的索引也可能会让性能下降至 1/100，因此在一个表中创建什么样的索引需要平衡各种业务需求。

对于数据库管理员来说，针对不同的业务特点，需要创建适合的索引，比较常见的索引有 B-TREE 索引、位图索引、全文索引。位图索引一般用于数据仓库应用，全文索引由于使用较

少，这里不深入介绍。

B-TREE 索引包括很多扩展类型，如组合索引、反向索引、函数索引等。B-TREE 索引也称为平衡树索引（Balance Tree），是一种按字段排好序的树形目录结构，主要用于提升查询性能和唯一约束支持。B-TREE 索引的内容包括根节点、分支节点、叶子节点。叶子节点存储索引字段内容和表记录 ROWID，根节点与分支节点保存了索引树的顺序及各层级间的引用关系，当一个数据块中不能放下所有索引字段数据时，就会形成树的根节点或分支节点。一个普通的 B-TREE 索引结构如图 7-3 所示。

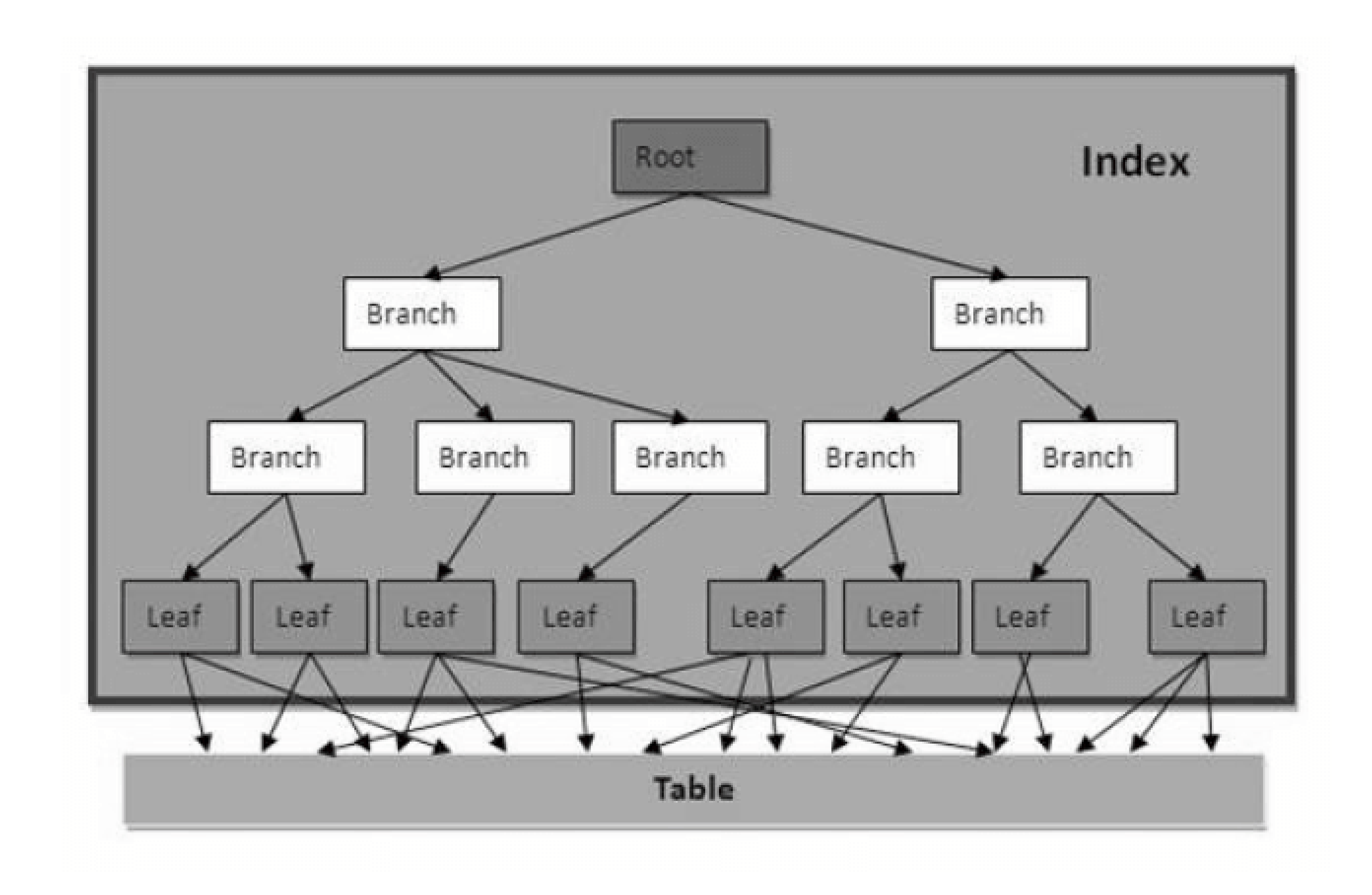


图 7-3　索引结构

上文介绍了最常用的索引结构，那么具体到操作层面，SQL 在什么条件下会使用索引呢？当字段上建有索引时，通常会在以下情况下使用索引：

- INDEX_COLUMN = ?
- INDEX_COLUMN > ?
- INDEX_COLUMN >= ?
- INDEX_COLUMN < ?
- INDEX_COLUMN <= ?
- INDEX_COLUMN between ? and ?
- INDEX_COLUMN in (?,?,…,?)
- INDEX_COLUMN like ?||'%'（后导模糊查询）
- T1. INDEX_COLUMN=T2. COLUMN1（两个表通过索引字段关联）

反过来看，SQL 不会使用索引的情况，见表 7-1。

表 7-1　不会使用索引的情况

<table>
<tr><th>查询条件</th><th>不能使用索引原因</th></tr>
<tr><td>INDEX_COLUMN <> ?
INDEX_COLUMN not in (?,?,…,?)</td><td>不等于操作不能使用索引</td></tr>
<tr><td>function(INDEX_COLUMN) = ?
INDEX_COLUMN + 1 = ?
INDEX_COLUMN || 'a' = ?</td><td>经过普通运算或函数运算后的索引字段不能使用索引</td></tr>
<tr><td>INDEX_COLUMN like '%'||?
INDEX_COLUMN like '%'||?||'%'</td><td>含前导模糊查询的 Like 语法不能使用索引</td></tr>
<tr><td>INDEX_COLUMN is null</td><td>B-TREE 索引里不保存字段为 NULL 值记录，因此 is null 不能使用索引</td></tr>
<tr><td>NUMBER_INDEX_COLUMN='12345'
CHAR_INDEX_COLUMN=12345</td><td>Oracle 在做数值比较时需要将两边的数据转换成同一种数据类型，如果两边数据类型不同就会对字段值隐式转换，相当于加了一层函数处理，所以不能使用索引</td></tr>
<tr><td>a.INDEX_COLUMN=a.COLUMN_1</td><td>给索引查询的值应是已知数据，不能是未知字段值</td></tr>
</table>

注：经过函数运算字段的字段要使用时可以用函数索引，这种需求建议与 DBA 沟通。

有时候会使用多个字段的组合索引，如果查询条件中第一个字段不能使用索引，那么整个查询也不能使用索引。例如，company 表建了一个 id+name 的组合索引，以下 SQL 是不能使用索引的：

```
Select * from company where name=?
```

Oracle 9i 后引入了一种 index skip scan 的索引方式来解决类似的问题，但是通过 index skip scan 提高性能的条件比较特殊，使用不好反而会使性能更差。

一般在什么字段上建索引，这是一个非常复杂的话题，需要对业务及数据充分分析后才能得出结果。主键及外键通常都要有索引，其他需要建索引的字段应满足以下条件：

（1）字段出现在查询条件中，并且查询条件可以使用索引。
（2）语句执行频率高，一天会有几千次以上。

通过字段条件可筛选的记录集很小，那数据筛选比例是多少才合适？这个没有固定值，需要根据表数据量来评估，以下是经验公式，可用于快速评估：

- 小表（记录数小于 10000 行的表）：筛选比例<10%。
- 大表：（筛选返回记录数）<（表总记录数*单条记录长度）/10000/16。
- 单条记录长度≈字段平均内容长度之和+字段数*2。

如何知道 SQL 是否使用了正确的索引？简单 SQL 可以根据索引使用语法规则判断，复杂的 SQL 不好办。判断 SQL 的响应时间是一种策略，但是这会受到数据量、主机负载及缓存等因素的影响，有时数据全在缓存里，可能全表访问的时间比索引访问时间还少。要准确知道索引是否正确使用，需要到数据库中查看 SQL 真实的执行计划，这个话题比较复杂，详见 SQL

执行计划专题介绍。

索引对 DML（Insert，Update，Delete）附加的开销有多少？这个没有固定的比例，与每个表记录的大小及索引字段大小密切相关，以下是一个普通表测试数据，仅供参考：

- 索引对于 Insert 性能降低 56%。
- 索引对于 Update 性能降低 47%。
- 索引对于 Delete 性能降低 29%。

因此对于写 IO 压力比较大的系统，表的索引需要仔细评估必要性，另外索引也会占用一定的存储空间。

有些时候，只是访问表中的几个字段，并且字段内容较少，可以为这几个字段单独建立一个组合索引，这样就可以直接只通过访问索引得到数据，一般索引占用的磁盘空间比表小得多，所以这种方式可以大大减少磁盘 IO 开销。

```
select id,name from company where type='2';
```

如果这个 SQL 经常使用，可以在 type、id、name 上创建组合索引：

```
create index my_comb_index on company(type,id,name);
```

有了这个组合索引后，SQL 就可以直接通过 my_comb_index 索引返回数据，不需要访问 company 表。还是拿字典举例：有一个需求，需要查询一本汉语字典中所有汉字的个数，如果字典没有目录索引，就只能从字典内容里一个一个地计数，最后返回结果。如果有一个拼音目录，就可以只访问拼音目录的汉字进行计数。如果一本字典有 1000 页，拼音目录有 20 页，那么数据访问成本就相当于全表访问的 1/50。

切记，性能优化是无止境的，性能满足需求即可，不要过度优化。在实际数据库中不可能把每个 SQL 请求的字段都建在索引里，所以这种只通过索引访问数据的方法一般只用于核心应用，也就是那种对核心表访问量较高且查询字段数据量很少的查询。SQL 执行计划是关系型数据库最核心的技术之一，表示 SQL 执行时的数据访问算法。由于业务需求越来越复杂，表数据量也越来越大，程序员越来越懒惰，SQL 也需要支持非常复杂的业务逻辑，但 SQL 的性能还需要提高，因此，优秀的关系型数据库除了需要支持复杂的 SQL 语法及更多函数外，还需要有一套优秀的算法库来提高 SQL 性能。

目前 ORACLE 有 SQL 执行计划的算法约 300 种，而且一直在增加，所以 SQL 执行计划是一个非常复杂的课题。一个普通 DBA 能掌握 50 种就很不错了，就算是资深 DBA 也不可能把每个执行计划的算法描述清楚。虽然有这么多种算法，但并不表示无法优化执行计划，因为常用的 SQL 执行计划算法也就十几个。如果一个程序员能把这十几个算法搞清楚，就掌握了 80%的 SQL 执行计划调优知识。

2. IO 优化的基本概念

首先向读者介绍 Oracle 数据库 I/O 相关竞争等待的相关内容。当 Oracle 数据库出现 I/O 相关的竞争等待时，一般来说都会引起 Oracle 数据库的性能低下，发现数据库存在 I/O 相关的

竞争等待。一般可以通过以下三种方法来查看 Oracle 数据库是否存在 I/O 相关的竞争等待：

- Statpack 报告中在"Top 5 Wait Events"部分中主要都是 I/O 相关的等待事件。
- 数据库等待事件的 SQL 语句跟踪中主要都是 I/O 相关的等待事件的限制。
- 操作系统工具显示存储数据库文件的存储磁盘有非常高的利用率。

如果发现数据库存在 I/O 竞争，就必须通过各种方法来调整优化 Oracle 数据库。在调优数据库的过程中，其中一个重要的步骤就是对响应时间的分析，看看数据库消耗的时间究竟消耗在什么上面了。对于 Oracle 数据库来说，响应时间的分析可以用下面的公式来计算：

Response Time = Service Time + Wait Time

- Service Time 是指'CPU used by this session'的统计时间。
- Wait Time 是指所有消耗在等待事件上的总时间。

如果使用性能调整的工具（如 statpack）来调整数据库，那么评测的则是所有响应时间中各个部分的相对影响，并且应该根据消耗的时间多少来调整影响最严重的部分。因为等待事件很多，因此还需要去判定哪些是真的很重要的等待事件。很多调优工具比如说 statpack 都会列出最重要的等待事件，statpack 工具报告中的重要等待事件都是包含在一个叫 Top 5 Wait Events 的部分中。因为这些工具都已经把重要的等待事件全部列出来了，所以就会处理这些已经列出来的等待事件而不必再去首先评估所有响应时间的影响。

在某些情况下， Service Time 会比 Wait Time 显得更加重要（例如 CPU 使用率），此时等待事件产生的影响就显得不是那么重要了，重点调整的目标应该放在 Service Time 上。因此，应该先比较在 Top 5 Wait Events 部分中的'CPU used by this session'所占用的时间，然后直接调整最消耗时间的等待事件。在 Oracle 9i 的 release 2 的版本以后，Top 5 Wait Events 部分变成了 Top 5 Timed Events，Service Time 也由'CPU used by this session'变成了'CPU time'来衡量，这也就意味着可以更加精确地判断在响应时间中等待事件的影响，从而调整最需要优化的部分。

下面具体说明为什么在调整数据库性能时必须同时查看 Service Time 和 Wait Time，因为如果不同时都查看这两个方面，就往往容易走入调整的误区。

```
Top5WaitEvents
~~~~~~~~~~~~~~~~~
Total Event                      Waits   Time (cs)   Wt Time
-------------------------------------------- ------------ ------------
-------
direct path read                     9，590      15，546    86。10
db file scattered read                  6，105      1，262     6。99
latch free                         2，036      1，047     5。80
log file sync                       107         131       。73
db file parallel write                  40          69        。38
```

上面是一个大约 30 分钟的 statpack 收集的信息的 Top 5 Wait Events 部分，如果基于上面给出的列表，很容易发现 direct path read 的 wait 很高，并且会试图去调整这个等待事件，但是这样做就没有考虑到 Service Time。下面来看看在这个 statpack 中关于 Service Time 的统计：

```
Statistic     Total   per Second    per Trans
--------------------------------- ---------------- ------------ ------------
CPU used by this session    429，648         238。7        257。4
```

下面大致计算一下响应时间：

```
'Wait Time' = 15，546 x 100% / 86。10% = 18，056 cs
'Service Time' = 429，648 cs
'Response Time' = 429，648 + 18，056 = 44，7704 cs
```

接着计算一下响应时间中各个部分的比例：

```
CPU time                     = 95。97%
direct path read               =  3。47%
db file scattered read          =  0。28%
latch free                    =  0。23%
log file sync                  =  0。03%
db file parallel write          =  0。02%
```

从上面的计算中可以明显地看出来，I/O 相关的等待事件所消耗的时间在整个响应时间中占的比例并不大，只不过是很小的一部分，相对来说 Service Time 所消耗的时间远远大于 Wait Time，因此，应该直接调整的是 Service Time（CPU 的使用率）而不是 I/O 相关的等待事件，在调优数据库时要尽量避免走入这种误区。

3. 处理 I/O 的方法

接着来具体看看对于出现的 I/O 问题处理的一些方法。

在使用 statpack 之类的工具分析了数据库的响应时间后，如果数据库的性能主要是被一些 I/O 相关的等待事件所限制住了，那么可以针对这种情况采用处理 I/O 问题的一些方法。下面对这些方法的一些概念和基本原理进行简单的阐述。

方法一：优化 Oracle 数据库的 SQL 语句来减少数据库对 I/O 的需求。如果数据库没有任何用户的 SQL 运行的话，一般来说只会产生很少的磁盘 I/O 或者几乎没有磁盘 I/O，基本上来说数据库产生 I/O 的最终原因都是直接或者间接地由于用户执行 SQL 语句导致的。这也就意味着可以控制单个 SQL 语句避免其产生大量的 I/O 来减少整个数据库对磁盘 I/O 的需求，通过优化 SQL 语句改变其执行计划以便让其产生尽可能少的 I/O。一般典型的存在问题的情况仅仅只是很少的几个 SQL 语句，但是由于其相应的执行计划不理想，会导致产生大量的物理磁盘 I/O，从而使得整个数据库的性能非常差。因此，让用户执行的 SQL 语句优化产生比较好的执行计划来减少磁盘 I/O 是一种行之有效的方法。

方法二：调整实例的初始化参数来减少数据库的 I/O 需求。一般来说可以通过两种途径实现。一种途径是通过内存缓存来减少 I/O。数据库的 I/O 分为两种，一种是实际读取了数据文件的物理 I/O，一种是从缓存中读取数据的逻辑 I/O，可以通过使用一定数量的内存缓存来减少物理 I/O（例如高速缓存区、日志缓存区以及各种排序区等）。适当地增大高速缓存区，可以有更多的缓存供给数据库的进程使用，从缓存中读取所需要的数据，这样产生的 I/O 都是逻辑 I/O，而不是直接从物理磁盘上读取数据，减少了物理 I/O 的产生。设置一个适当的排序区，

可以减少在排序操作中读取临时表空间数据文件所在磁盘的次数，而尽可能多地使用缓存中的排序区来排序。其他的缓存区的工作原理基本都是一致的，都是通过使用缓存来减少读取物理磁盘的次数来降低 I/O。另外一种途径是调整一次读取多个 BLOCK 的大小，单独的一次多个 BLOCK 读取的操作的大小是由实例的初始化参数 db_file_multiblock_read_count 来控制的。如果执行比较大的 I/O 操作，一次读取的多个 BLOCK 大小越大，所需要的时间就会越短。例如，操作系统一次能够传输的最大 I/O 大小是 8MB，一次性请求传输 50MB 的数据，会比请求 5 次且每次只是请求 10MB 的速度快得多。但是，当一次读取的多个 BLOCK 的大小超过了操作系统一次能够传输的最大的 I/O 大小时，这个差别就基本不明显了。如一次性传输 100MB 和一次性传输 1GB 的大小在时间上基本没有什么差别了，效率差不多。因为整个 I/O 所消耗的时间分为 I/O Setup Time 和 I/O Transfer Time 两个主要部分，I/O Setup Time 可以看成是 I/O 的寻道所消耗的时间，I/O Transfer Time 可以看成是 I/O 传输所消耗的时间。当一次读取的多个 BLOCK 的大小比较小时，读取一定数量的 BLOCK 就会使得读取次数比较多，每次 I/O 读取都要先寻道，并且寻道时间在这个时候所用的时间会占到整个 I/O 完成时间的绝大部分，导致整个 I/O 完成所消耗的时间也就会比较多了。而 I/O 传输一定数量的 BLOCK 的时间相对固定，不管传输的次数多少，基本上变化不大，因此 I/O 读取时间长短主要取决于 I/O Setup Time 所花费的时间。因此，在配置数据库初始化参数时，根据操作系统的 I/O 吞吐能力都会设置一次读取多个 BLOCK 的大小尽量多，以减少读取 I/O 的次数。

方法三：在操作系统级别上优化 I/O，在操作系统级别上优化磁盘的 I/O，以提高 I/O 的吞吐量，如果操作系统支持异步 I/O，尽量去使用异步 I/O；还可以使用高级文件系统的一些特性，例如直接 I/O 读取，忽略操作系统的文件缓存，也就是平时所说的使用裸设备；还有一种可行的方法是增大每次传输的最大 I/O 大小的限制，以便使每次能够传输的 I/O 尽可能大。

方法四：通过使用 RAID、SAN、NAS 来平衡数据库的 I/O。

① RAID 是 Redundent Array of Independent Disks 的缩写，直译为“廉价冗余磁盘阵列”，也简称为“磁盘阵列”。RAID 的优点是传输速率高并且可以提供容错功能。在 RAID 中，可以让很多磁盘驱动器同时传输数据，而这些磁盘驱动器在逻辑上又是一个磁盘驱动器，所以使用 RAID 可以达到单个磁盘驱动器几倍、几十倍甚至上百倍的速率。因为普通磁盘驱动器无法提供容错功能，如果不包括写在磁盘上的 CRC（循环冗余校验）码的话。RAID 容错是建立在每个磁盘驱动器的硬件容错功能之上的，可以提供更高的安全性。RAID 分为以下几个级别：

- RAID0:RAID0 并不是真正的 RAID 结构，没有数据冗余。RAID0 连续地分割数据并并行地读/写于多个磁盘上，因此具有很高的数据传输率。但 RAID0 在提高性能的同时，并没有提供数据可靠性，如果一个磁盘失效，将影响整个数据，因此 RAID0 不可应用于需要数据高可用性的关键应用。
- RAID1:RAID1 通过数据镜像实现数据冗余，在两对分离的磁盘上产生互为备份的数据。RAID1 可以提高读的性能，当原始数据繁忙时，可直接从镜像备份中读取数据。RAID1 是磁盘阵列中费用最高的，但提供了最高的数据可用率。当一个磁盘失效时，系统可以自动地交换到镜像磁盘上，而不需要重组失效的数据。

- RAID2:从概念上讲，RAID2 同 RAID3 类似，两者都是将数据条块化分布于不同的硬盘上，条块单位为位或字节。然而 RAID2 使用称为“加重平均纠错码”的编码技术来提供错误检查及恢复。这种编码技术需要多个磁盘存放检查及恢复信息，使得 RAID2 技术实施更复杂。因此，在商业环境中很少使用。
- RAID3:不同于 RAID2，RAID3 使用单块磁盘存放奇偶校验信息。如果一块磁盘失效，奇偶盘及其他数据盘可以重新产生数据。如果奇偶盘失效，就不会影响数据使用。RAID3 对于大量的连续数据可提供很好的传输率，但对于随机数据，奇偶盘会成为写操作的瓶颈。
- RAID4:同 RAID2、RAID3 一样，RAID4 和 RAID5 也将数据条块化并分布于不同的磁盘上，但条块单位为块或记录。RAID4 使用一块磁盘作为奇偶校验盘，每次写操作都需要访问奇偶盘，成为写操作的瓶颈。因此，在商业应用中很少使用。
- RAID5:RAID5 没有单独指定的奇偶盘，而是交叉地存取数据及奇偶校验信息于所有磁盘上。在 RAID5 上，读/写指针可同时对阵列设备进行操作，提供了更高的数据流量。RAID5 更适合于小数据块、随机读写的数据。RAID5 与 RAID3 相比，重要的区别在于 RAID3 每进行一次数据传输，需涉及所有的阵列盘；而对于 RAID5 来说，大部分数据传输只对一块磁盘操作，可进行并行操作。在 RAID5 中有“写损失”，即每一次写操作将产生四个实际的读/写操作，其中两次读旧的数据及奇偶信息、两次写新的数据及奇偶信息。
- RAID6:RAID6 与 RAID5 相比，增加了第二个独立的奇偶校验信息块。两个独立的奇偶系统使用不同的算法，数据的可靠性非常高。即使两块磁盘同时失效，也不会影响数据的使用。但需要分配给奇偶校验信息更大的磁盘空间，相对于 RAID5 有更大的“写损失”。RAID6 的写性能非常差。较差的性能和复杂的实施使得 RAID6 很少使用。

② SAN（Storage Area Network，存储局域网）是独立于服务器网络系统之外几乎拥有无限存储能力的高速存储网络。这种网络采用高速的光纤通道作为传输媒体，以 FC （Fiber Channel，光通道）+ SCSI（Small Computer System Interface，小型计算机系统接口） 的应用协议作为存储访问协议，将存储子系统网络化，实现了真正高速共享存储的目标。一个完整的 SAN 包括支持 SAN 的主机设备、支持 SAN 的储存设备、用于连接 SAN 的连接设备、支持 SAN 的管理软件、支持 SAN 的服务。

③ NAS（Network Attached Storage，网络附加存储设备）是一种专业的网络文件存储及文件备份设备，或称为网络直联存储设备、网络磁盘阵列。NAS 是基于 LAN 的，按照 TCP/IP 协议进行通信，面向消息传递，以文件的 I/O 方式进行数据传输。在 LAN 环境下，NAS 已经完全可以实现异构平台之间的数据级共享，比如 NT、UNIX 等平台的共享。一个 NAS 包括处理器、文件服务管理模块和多个硬盘驱动器，用于数据的存储。 NAS 可以应用在任何网络环境当中。主服务器和客户端可以非常方便地在 NAS 上存取任意格式的文件，包括 SMB 格式（Windows）、NFS 格式（UNIX， Linux）和 CIFS 格式等。NAS 系统可以根据服务器或

者客户端计算机发出的指令完成对内在文件的管理。NAS 是在 RAID 的基础上增加了存储操作系统，因此，NAS 的数据能由异类平台共享。

因此，利用 RAID、SAN、NAS 的技术在多个物理磁盘之间平衡数据库的 I/O，尽量避免数据库产生 I/O 竞争的瓶颈。

方法五：手动分配数据文件到不同的文件系统、控制器和物理设备来重新调整数据库 I/O。如果数据库目前的存储设备不算太好，那么采用这种方法是一个不错的选择。这样可以让所有的磁盘得到充分的利用，不至于出现某些磁盘的 I/O 太高、某些磁盘根本没有被使用的情况，在配置较低的情况下得到一个比较好的数据库性能。需要注意的一点是对于大部分数据库来说，一些 I/O 是一直会存在的。如果上述的方法都尝试过但是数据库的 I/O 性能还是没有达到预定的要求，可以尝试删除数据库中一些不用的旧数据或者使用性能更好的硬件设施。

4. 等待事件

下面总结在 Oracle 数据库中经常出现的一些 I/O 相关的等待事件。

- 数据文件 I/O 相关的等待事件
  - db file sequential read
  - db file scattered read
  - db file parallel read
  - direct path read
  - direct path write
  - direct path read (lob)
  - direct path write (lob)
- 控制文件 I/O 相关的等待事件
  - control file parallel write
  - control file sequential read
  - control file single write
- 重做日志文件 I/O 相关的等待事件
  - log file parallel write
  - log file sync
  - log file sequential read
  - log file single write
  - switch logfile command
  - log file switch completion
  - log file switch (clearing log file)
  - log file switch (checkpoint incomplete)
  - log switch/archive
  - log file switch (archiving needed)

- 高速缓存区 I/O 相关的等待事件
  - db file parallel write
  - db file single write
  - write complete waits
  - free buffer waits

（1）db file sequential read 等待事件

这个是非常常见的 I/O 相关的等待事件。在大多数的情况下读取一个索引数据的 BLOCK 或者通过索引读取数据的一个 BLOCK 时都要读取相应的数据文件头的 BLOCK。在早期的版本中会从磁盘中的排序段读取多个 BLOCK 到高速缓存区的连续缓存中。

在 V$SESSION_WAIT 视图里面，这个等待事件有三个参数 P1、P2、P3，其中 P1 代表 Oracle 要读取的文件的 ABSOLUTE 文件号，P2 代表 Oracle 从这个文件中开始读取的 BLOCK 号，P3 代表 Oracle 从这个文件开始读取的 BLOCK 号后读取的 BLOCK 数量，通常这个值为 1，表明是单个 BLOCK 被读取，如果这个值大于 1，就是读取了多个 BLOCK，这种多 BLOCK 读取常常出现在早期的 Oracle 版本从临时段中读取数据时。

如果这个等待事件在整个等待时间中占主要的部分，就可以采用以下几种方法来调整数据库。

方法一：从 statpack 报告中的 SQL ordered by Reads 部分或者从 V$SQL 视图中找出读取物理磁盘 I/O 最多的几个 SQL 语句，优化这些 SQL 语句以减少对 I/O 的读取需求。

如果有 Index Range scans，但是却使用了不该用的索引，就会导致访问更多的 BLOCK，这时应该强迫使用一个可选择的索引，使访问同样的数据尽可能少访问索引块，减少物理 I/O 的读取；如果索引的碎片比较多，那么每个 BLOCK 存储的索引数据就比较少，这样需要访问的 BLOCK 就多，这时一般来说最好把索引 rebuild，减少索引的碎片；如果被使用的索引存在一个很大的 Clustering Factor，那么对于每个索引 BLOCK 获取相应的记录时就要访问更多表的 BLOCK，这时可以使用特殊的索引列排序来重建表的所有记录，这样可以大大地减少 Clustering Factor。例如，一个表有 A、B、C、D、E 五列，索引建立在 A、C 上，可以使用如下语句来重建表：

```
CREATE TABLE TABLE_NAME AS SELECT * FROM old ORDER BY A,C;
```

此外，还可以通过使用分区索引来减少索引 BLOCK 和表 BLOCK 的读取。

方法二：如果不存在有问题的执行计划导致读取过多的物理 I/O 的特殊 SQL 语句，那么数据文件所在的磁盘可能存在大量的活动，导致其 I/O 性能很差，这种情况下可以通过查看 Statpack 报告中的“File I/O Statistics”部分或者 V$FILESTAT 视图找出热点的磁盘，将在这些磁盘上的数据文件移动到那些使用了条带集、RAID 等能实现 I/O 负载均衡的磁盘上去。

从 Oracle 9.2.0 开始，可以从 V$SEGMENT_STATISTICS 视图中找出物理读取最多的索引段或者表段，通过查看这些数据，可以清楚详细地看到这些段是否可以使用重建或者分区的方法来减少所使用的 I/O。如果 Statpack 设置的 level 为 7 就会在报告中产生 Segment Statistics 信息。

```
SQL> select distinct statistic_name from v$segment_statistics;
STATISTIC_NAME
----------------------------------------------------------------
ITL waits
buffer busy waits
db block changes
global cache cr blocks served
global cache current blocks served
logical reads
physical reads
physical reads direct
physical writes
physical writes direct
row lock waits
11 rows selected。
```

从上面的查询可以看到相应的统计名称，使用下面的查询语句就能得到读取物理 I/O 最多的段：

```
select object_name,object_type,statistic_name,value
from v$segment_statistics
where statistic name='physical reads'
order by value desc;
```

方法三：如果不存在有问题的执行计划导致读取过多的物理 I/O 的特殊 SQL 语句，磁盘的 I/O 也分布得很均匀，这时可以考虑增大高速缓存区。对于 Oracle 8i 来说，增大初始化参数 DB_BLOCK_BUFFERS，让 Statpack 中的 Buffer Cache 的命中率达到一个满意值；对于 Oracle 9i 来说，可以使用 Buffer Cache Advisory 工具来调整 Buffer Cache；对于热点的段，可以使用多缓冲池，将热点的索引和表放入 KEEP Buffer Pool 中去，尽量让其在缓冲中被读取，减少 I/O。

（2）db file scattered read 等待事件

这也是一个非常常见的等待事件。当 Oracle 从磁盘上读取多个 BLOCK 到不连续的高速缓存区的缓存中时就会发生这个等待事件。Oracle 一次能够读取的最多的 BLOCK 数量是由初始化参数 DB_FILE_MULTIBLOCK_READ_COUNT 来决定的，这个等待事件一般伴随着全表扫描或者 Fast Full Index 扫描一起出现。

在 V$SESSION_WAIT 视图里面，这个等待事件有三个参数 P1、P2、P3。其中，P1 代表 Oracle 要读取的文件的 ABSOLUTE 文件号，P2 代表 Oracle 从这个文件中开始读取的 BLOCK 号，P3 代表 Oracle 从这个文件开始读取的 BLOCK 号后读取的 BLOCK 数量。

如果这个等待事件在整个等待时间中占了比较大的比重，可以采用如下几种方法来调整 Oracle 数据库。

方法一：找出执行全表扫描者 Fast Full Index 扫描的 SQL 语句，判断这些扫描是否是必要的，是否导致了比较差的执行计划，如果是，就需要调整这些 SQL 语句。

从 Oracle 9i 开始提供了一个视图 V$SQL_PLAN，可以很快地帮助找到那些全表扫描或者

Fast Full Index 扫描的 SQL 语句。这个视图会自动忽略关于数据字典的 SQL 语句。

查找全表扫描的 SQL 语句可以使用如下语句：

```
select sql_text from v$sqltext t, v$sql_plan p
where t。hash_value=p。hash_value and p。operation='TABLE ACCESS'
and p。options='FULL'
order by p。hash_value, t。piece;
```

查找 Fast Full Index 扫描的 SQL 语句可以使用如下语句：

```
select sql text from v$sqltext t, v$sql plan p
where t。hash value=p。hash value and p。operation='INDEX'
and p。options='FULL SCAN'
order by p。hash_value, t。piece;
```

如果是 Oracle 8i 的数据库，可以从 V$SESSION_EVENT 视图中找到关于这个等待事件的进程 sid，然后根据 sid 来跟踪相应的会话的 SQL。

```
select sid，event from v$session_event where event='db file sequential read'
```

或者可以查看物理读取最多的 SQL 语句的执行计划，看是否里面包含了全表扫描和 Fast Full Index 扫描。可以通过如下语句来查找物理读取最多的 SQL 语句：

```
select sql_text from (
select * from v$sqlarea
order by disk reads)
where rownum<=10;
```

方法二：有时候在执行计划很好的情况下也会出现多 BLOCK 扫描的情况，这时可以通过调整 Oracle 数据库的多 BLOCK 的 I/O，设置一个合理的 Oracle 初始化参数 DB_FILE_MULTIBLOCK_READ_COUNT，尽量满足以下公式：

DB_BLOCK_SIZE x DB_FILE_MULTIBLOCK_READ_COUNT= max_io_size of system

DB_FILE_MULTIBLOCK_READ_COUNT 是指在全表扫描中一次能够读取的最多的 BLOCK 数量，这个值受操作系统每次能够读写最大的 I/O 限制，如果设置的值按照上面的公式计算超过了操作系统每次的最大读写能力，则会默认为 max_io_size/db_block_size。例如 DB_FILE_MULTIBLOCK_READ_COUNT 设置为 32，DB_BLOCK_SIZE 为 8KB，这样每次全表扫描时就能读取 256KB 的表数据，从而大大地提高了整体查询的性能。设置这个参数也不是越大越好，设置这个参数之前应该先了解应用的类型，如果是 OLTP 类型的应用，一般来说全表扫描较少，这时设定比较大的 DB_FILE_MULTIBLOCK_READ_COUNT 反而会降低 Oracle 数据库的性能，因此 CBO 在某些情况下会因为多 BLOCK 读取导致 COST 比较低，从而错误地选用全表扫描。此外，还可以通过对表和索引使用分区、将缓存区的 LRU 末端的全表扫描和 Fast Full Index 扫描的 BLOCK 放入 KEEP 缓存池中等方法调整这个等待事件。

（3）db file parallel read 等待事件

当 Oracle 从多个数据文件中并行读取多个 BLOCK 到内存的不连续缓冲中（高速缓存区

或者是 PGA）时可能就会出现这个等待事件。这种并行读取一般出现在恢复操作或者从缓冲中预取数据达到最优化（而不是多次从单个 BLOCK 中读取）。这个事件表明会话正在并行执行多个读取的需求。在 V$SESSION_WAIT 视图里面，这个等待事件有三个参数 P1、P2、P3。其中，P1 代表有多少个文件被读取所请求，P2 代表总共有多少个 BLOCK 被请求，P3 代表总共有多少次请求。如果在等待时间中这个等待事件占的比重比较大，就可以按照处理 db file sequential read 等待事件的方法来处理这个事件。

（4）direct path read/write 等待事件

这个等待事件一般出现在 Oracle 将数据直接读入或写入到 PGA 内存中（而不是高速缓存区），如果系统使用了异步 I/O，那么 Oracle 可以一边提交请求一边同时继续处理请求，这样能加速 I/O 请求的结果并会出现 direct path read 等待直到请求 I/O 完成。如果没有使用异步 I/O，I/O 请求会被阻塞直到之前的 I/O 请求完成后，但是此时不会出现 I/O 等待，会话稍后重新恢复并加速 I/O 请求的完成，此时就会出现 direct path read/write 等待。因此，对于这个等待事件容易产生两方面的误解：一是认为等待的总数量不能反映出 I/O 请求的数量，二是消耗在这个等待事件上的总时间不能反映出实际的等待时间。这种类型的读取请求主要是用于不在内存中排序的 I/O、并行查询以及预读操作（提前请求一个进程即将使用的 BLOCK）。

在 V$SESSION_WAIT 视图里面，这个等待事件有三个参数 P1、P2、P3。其中，P1 代表等待 I/O 读取请求的文件的 ABSOLUTE 文件号，P2 代表等待 I/O 读取请求的第一个 BLOCK 号，P3 代表总共有多少个连续的 BLOCK 被请求读取。

这个等待事件的等待时间是指等待 BLOCK 直到显著的 I/O 请求完成的时间。值得注意的是，对于异步 I/O 来说等待时间并不是 I/O 本身所耗费的时间，因为此时等待并没有开始，而且在这个等待事件中 Oracle 并不会出现超时的现象。在 DSS 类型的系统中，执行大量批处理操作的过程中出现这个等待事件属于很正常的现象，然而如果在 OLTP 类型的系统中大量出现这个等待事件则表明 Oracle 数据库存在问题需要调整。如果在等待时间中这个等待事件占的比重比较大，可以从如下几方面来调整。如果等待的文件是临时表空间的数据文件，那么需要查看是否存在大量不合理的磁盘排序，优化相应的存在问题的 SQL 语句。另外建议确认异步 I/O 是否配置正确，使用异步 I/O 不会减少这个等待事件的等待时间，却可以减少会话所消耗的时间。检查是否存在 I/O 消耗很严重的 SQL 语句，如果存在，尝试优化 SQL 语句，减少 I/O 的消耗。最后确认一下是否达到了磁盘的 I/O 极限，如果是，就需要考虑更换性能更好的硬件设备。

（5）direct path read/write (lob) 等待事件

这个等待事件是从 Oracle 8.1.7 开始出现的，表明在等待直接路径读取访问一个 LOB 对象。在 V$SESSION_WAIT 视图里面，这个等待事件有三个参数 P1、P2、P3。其中，P1 代表等待 I/O 读取请求的文件的 ABSOLUTE 文件号，P2 代表等待 I/O 读取请求的第一个数据 BLOCK 地址，P3 代表总共有多少个连续的 BLOCK 被请求读取。对于那些没有 cache 的 LOB 对象，强烈建议将其所在的数据文件放置在存在缓存的磁盘上（例如文件系统），这样使得直接读取操作能够受益于那些非 Oracle 的 cache，加快读取的速度。

（6）control file parallel write 等待事件

这个等待事件表明服务器进程在更新所有的控制文件时等待 I/O 的完成。因为控制文件所在的磁盘的 I/O 过高引起无法完成对所有控制文件的物理写入，写入控制文件的这个会话会拥有 CF 队列，因此其他的会话都会在这个队列中等待。

在 V$SESSION_WAIT 视图里面，这个等待事件有三个参数 P1、P2、P3，这三个参数都设置为同样的值，代表控制文件对 I/O 的请求数量。当 Oracle 更新控制文件时同时更新所有控制文件并写入同样的信息。

如果在等待时间中这个等待事件占的比重比较大，可以从如下几方面来调整。

- 在确保控制文件不会同时都丢失的前提下，将控制文件的数量减小到最少。
- 如果系统支持异步 I/O，则推荐使用异步 I/O，这样可以实现真正并行的写入控制文件。
- 将控制文件移动到负载比较低、速度比较快的磁盘上。

（7）control file sequential read 等待事件

读取控制文件时遇到 I/O 等待就会出现这个等待事件，例如备份控制文件时、读取 BLOCK 头部等都会引起这个等待事件，等待的时间就是消耗在读取控制文件上的时间。

在 V$SESSION_WAIT 视图里面，这个等待事件有三个参数 P1、P2、P3。其中，P1 代表正在读取的控制文件号，通过下面的 SQL 语句可以知道究竟是具体是哪个控制文被读取：

```
SELECT * FROM X$KCCCF WHERE INDX = <file#>;
```

P2 代表开始读取的控制文件 BLOCK 号，它的 BLOCK 大小和操作系统的 BLOCK 大小一样，通常来说是 512KB，也有些 UNIX 的是 1MB 或者 2MB。P3 代表会话要读取 BLOCK 的数量。一般使用参数 P1、P2 来查询 BLOCK，当然也可以包括参数 P3，但是那样最终就变成了一个多 BLOCK 读取，因此一般都忽略参数 P3。

如果这个等待事件等待的时间比较长，则需要检查控制文件所在的磁盘是否很繁忙，如果是，就将控制文件移动到负载比较低、速度比较快的磁盘上。如果系统支持异步 I/O，就启用异步 I/O。对于并行服务器来说，如果这种等待比较多，会造成整个数据库性能下降，因为并行服务器之间的一些同步是通过控制文件来实现的。

（8）control file single write 等待事件

这个等待事件出现在写控制文件的共享信息到磁盘时，这是一个自动操作，并且通过一个实例来保护。如果是并行的数据库服务器，那么对于并行服务器来说也只能有一个实例能够执行这个操作。这个事件的等待时间就是写操作所消耗的时间。

在 V$SESSION_WAIT 视图里面，这个等待事件有三个参数 P1、P2、P3。其中，P1 代表正在读取的控制文件号，通过下面的 SQL 语句可以知道究竟是哪个控制文被读取：

```
SELECT * FROM X$KCCCF WHERE INDX = <file#>;
```

P2 代表开始读取的控制文件 BLOCK 号，它的 BLOCK 大小和操作系统的 BLOCK 大小一样，通常来说是 512KB，也有些 UNIX 的是 1KB 或者 2KB。P3 代表会话要读取 BLOCK 的数量。

一般使用参数 P1、P2 来查询 BLOCK，当然也可以包括参数 P3，但是那样最终就变成了一个多 BLOCK 读取，因此一般都忽略参数 P3。

尽管这个事件是 single write，事实上也会出现多 BLOCK 写的情况，即 P3>1。可以使用参数 P1、P2 来查询检测 BLOCK，而不用去考虑 P3 的值。

如果这个等待事件等待的时间比较长，就需要检查控制文件所在的磁盘是否很繁忙，如果是，就将控制文件移动到负载比较低、速度比较快的磁盘上。如果系统支持异步 I/O，就启用异步 I/O。对于并行服务器来说，如果这种等待比较多，就会造成整个数据库性能下降，因为并行服务器之间的一些同步是通过控制文件来实现的。

（9）log file parallel write 等待事件

这个等待事件出现在当 LGWR 后台进程从日志缓冲区写日志信息到磁盘上的重做日志文件时。只有启用了异步 I/O 时 LGWR 进程才会并行写当前日志组内的重做日志文件，否则 LGWR 只会循环顺序写当前日志组重做日志文件。LGWR 进程不得不等待当前日志组所有的重做日志文件成员全部写完，因此，决定这个等待事件等待时间长短的主要因素是重做日志文件所在磁盘的 I/O 读写速度。

如果是由于当前的 LGWR 进程写的速度不够快导致了这个等待事件，可以通过查看一些和重做日志相关的统计值判定当前的 LGWR 进程效率是否很低，具体的可以查看"redo writes" "redo blocks written" "redo write time" "redo wastage" "redo size"统计值，这些都是和 LGWR 进程性能直接相关的一些统计值。

在 V$SESSION_WAIT 视图里面，这个等待事件有三个参数 P1、P2、P3。其中，P1 代表正在被写入的重做日志文件组中的重做日志文件号，P2 代表需要写入重做日志组中每个重做日志文件的重做日志 BLOCK 数量，P3 代表 I/O 请求的次数，需要被写入的 BLOCK 会被分成多次分别请求。

如果这个等待事件占用的等待时间比较多，可以从以下几个方面来进行调整。

- 对于能使用 UNRECOVERABLE/NOLOGGING 的操作，尽量使用这两个选项来减少重做日志的产生。在保证不会同时丢失重做日志文件的前提下尽量减少重做日志组中的成员个数，减少每次写重做日志组文件的时间。
- 除非在备份的情况下，否则不要将表空间置于热备的模式下，因为表空间处于热备的模式下会产生更多的重做日志文件。
- 对于使用 LogMiner、Logical Standby 或者 Streams，在能够满足要求功能的前提下，尽量使用最低级别的追加日志，以减少重做日志的产生。
- 尽量将同一个日志组内的重做日志文件分散到不同的硬盘上，减少并行写重做日志文件时产生的 I/O 竞争。
- 不要将重做日志文件放置在 RAID-5 的磁盘上，最好使用裸设备来存放重做日志文件。
- 如果设置了归档模式，不要将归档日志的目的地设置为存放重做日志存放的磁盘上面，避免引起 I/O 竞争。

（10）log file sync 等待事件

这个等待事件是指等待 Oracle 的前台的 COMMIT 和 ROLLBACK 操作进程完成，有时这个等待事件也会包括等待 LGWR 进程把一个会话事务的日志记录信息从日志缓冲区中写入磁盘上的重做日志文件中。因此，当前台进程在等待这个事件时，LGWR 进程同时也在等待事件 log file parallel write。理解什么造成这个等待事件的关键在于对比这个等待事件和 log file parallel write 等待事件的平均等待时间：如果它们的等待时间差不多，就是重做日志文件的 I/O 引起了这个等待事件，则需要调整重做日志文件的 I/O，这个在之后会有详细的讲述。如果 log file parallel write 等待事件的平均等待时间明显小于 log file sync 等待事件的等待时间，那么就是一些其他的写日志的机制在 COMMIT 和 ROLLBACK 操作时引起了等待，而不是 I/O 引起的等待。例如，重做日志文件的 latch 的竞争，会伴随着出现 latch free 或者 LGWR wait for redo copy 等待事件。

在 V$SESSION_WAIT 视图里面，这个等待事件有三个参数 P1、P2、P3。其中，P1 代表在日志缓冲区中需要被写入到重做日志文件中的缓存的数量，写入的同时会确认事务是否已经被提交，并且保留提交信息到实例意外中断之前，因此必须等待 LGWR 将 P1 数量的缓存写入重做日志文件为止。P2、P3 属于无用的参数。

如果这个等待事件在整个等待时间中占了比较大的比重，可以从以下三个方面来调整这个等待事件。

- 调整 LGWR 进程使其具有更好的磁盘 I/O 吞吐量，例如不要将日志文件放置在 RAID5 的磁盘上。
- 如果存在很多执行时间很短的事务，可以考虑将这些事务集合成一个批处理事务，以减少提交的次数，因为每次提交都需要确认相关的日志写入重做日志文件。因此，使用批处理事务来减少提交的次数是一种非常行之有效的减少 I/O 的方法。
- 查看一些操作是否可以安全地使用 NOLOGGING 或者 UNRECOVERABLE 选项，以减少日志的产生。

（11）log file sequential read 等待事件

这个等待事件是指等待读取重做日志文件中的日志记录，等待的时间就是耗费在完成整个读取日志记录的物理 I/O 操作的时间。

在 V$SESSION_WAIT 视图里面，这个等待事件有三个参数 P1、P2、P3。其中，P1 代表一个日志组里面所有日志文件的相对 sequence 号，P2 代表日志文件在指定物理块大小的偏移量，P3 代表读取 BLOCK 的数量，如果 P3 的值为 1，一般来说都是在读取日志文件头。

（12）log file single write 等待事件

这个等待事件是指等待写重做日志文件操作完成，常常是在等待写重做日志文件头。例如，在增加一个新的重做日志组成员时，Oracle 数据库就会往这个重做日志文件头写入相应的 sequence 号。在 V$SESSION_WAIT 视图里面，这个等待事件有三个参数 P1、P2、P3。其中，P1 代表正在被写入的重做日志文件组的组号，P2 代表日志文件在指定物理块大小的偏移量，P3 代表写入 BLOCK 的数量。因为 single write 通常都是在写或者重写日志文件头时出现，因

此开始的 block 号总是为 1。一般如果出现这个等待事件，应该对重做日志文件尽量使用裸设备，避免将多个日志文件放在同一个磁盘上，减少产生 I/O 竞争的可能。

（13）switch logfile command 等待事件

这个等待事件是指执行日志文件切换命令时等待日志文件切换完成，Oracle 数据库会每隔五秒钟就检测一次是否超时。如果出现这个等待事件，表明花费了很长的时间去切换重做日志文件，此时需要去检查数据库的告警日志文件，查看 Oracle 后台进程 LGWR 是否在正常工作。

（14）log file switch completion 等待事件

这个等待事件是指由于当前重做日志文件已经被写满了而 Oracle 后台进程 LGWR 需要写完当前重做日志文件并且要打开一个新的重做日志文件而导致的重做日志文件切换的等待，或者是其他请求需要切换重做日志文件导致等待。

如果当前的重做日志写满了，这时 Oracle 数据库就需要切换重做日志文件来提供足够的磁盘空间给重做日志写日志缓存。由于一些其他的进程也同样可以引起重做日志的切换，Oracle 数据库不会同时去切换重做日志两次，因此就出现了这个等待事件。在 Oracle 数据库早期的版本中还有 log_file_switch_checkpoint_incomplete、log_file_switch_archiving_needed、log_file_switch_clearing_log_file 等待事件。

（15）log file switch (checkpoint incomplete) 等待事件

这个等待事件是指由于当前重做日志的检查点没有及时完成而导致重做日志文件无法切换到下一个日志文件引起的日志文件切换的等待。

调整这个等待事件的方法一般是加速检查点的完成，可以通过减小 buffer cache 缓冲区或者增加更多的 DBWR 进程、调整相关检查点的初始化参数等方法来达到相应的效果。

（16）log file switch (archiving needed)等待事件

这个等待事件是指当前的重做日志文件准备切换到下一个重做日志文件，但是当前重做日志文件因为没有被归档而导致等待，这个等待事件只出现于采用了归档方式的 Oracle 数据库中。

如果出现这个等待事件，首先应该查看 Oracle 数据库的告警日志文件，看是否因为写归档日志文件错误导致归档进程停止；其次，可以增加归档进程的数量或者将归档日志文件存放到 I/O 速度比较快的磁盘上；最后，还可以通过增大和增加重做日志文件的大小和数量来给予归档更多的时间。

（17）db file parallel write 等待事件

这个等待事件是指 Oracle 后台进程 DBWR 等待一个并行写入文件或者 BLOCK 的完成，等待会一直持续到这个并行写入操作完成。

在 V$SESSION_WAIT 视图里面，这个等待事件有三个参数 P1、P2、P3。其中，P1 代表 Oracle 正在写入的数据文件的数量，P2 代表操作将会写入多少的 BLOCK 数量，P3 在 Oracle 9i release2 版本之前代表总共有多少 BLOCK 的 I/O 请求，等于 P2 的值；在 Oracle 9i release2 版本之后则代表等待 I/O 完成的超时时间，单位是百分之一秒。

这个等待事件即使在总的等待时间中占的比例比较大也不会对用户的会话有很大的影响，只有当用户的会话显示存在大量的等待时间消耗在“write complete waits”或者是“free buffer waits”上时才会影响到用户的会话，较明显的影响是这个写操作的等待会影响到读取同一个磁盘上数据的用户会话的 I/O。

（18）db file single write 等待事件

这个等待事件通常是表明在等待写入数据到数据文件头。

在 V$SESSION_WAIT 视图里面，这个等待事件有三个参数 P1、P2、P3。其中，P1 代表 Oracle 正在写入的数据文件的文件号：

```
SELECT * FROM v$datafile WHERE file# = <file#>;
```

P2 代表 Oracle 正在写入的 BLOCK 号，如果 BLOCK 号不是 1，就可以通过如下查询查出 Oracle 正在写入的对象是什么：

```
select  segment_name ,  segment_type ,
owner , tablespace_name
from  sys。dba_extents
where  file_id = <file#>
and  <block#>
between block_id and block_id + blocks -1;
```

P3 代表 Oracle 写入 file＃的数据文件中从 BLOCK#开始写入的 BLOCK 的数量。

Oracle 数据文件的文件头一般都是 BLOCK1，操作系统指定的文件头是 BLOCK0，如果 BLOCK 号大于 1，则表明 Oracle 正在写入的是一个对象而不是文件头。

（19）write complete waits 等待事件

这个等待事件表明 Oracle 的会话在等待写入缓存，一般都是缓存的正常老化或者是实例之间互相调用引起的。

在 V$SESSION_WAIT 视图里面，这个等待事件有三个参数 P1、P2、P3。其中，P1 代表 Oracle 正在写入的数据文件的文件号，P2 代表 Oracle 正在写入的 BLOCK 号，可以通过如下查询查出 Oracle 正在写入的对象是什么：

```
select  segment_name ,  segment_type ,owner , tablespace_name
from  sys。dba_extents
where  file_id = <file#>
and  <block#> between block_id and block_id + blocks -1;
```

P3 代表产生这个等待事件原因的 id 号，具体的 id 号所代表的原因如下：

- 1022 无。
- 1027 在写入过程中的 buffer，最多等待 1 秒后会重新扫描 cache。
- 1029 试图应用改变到正在写入中的 BLOCK，会一直等到 BLOCK 可用，每次等待的时间都是 1 秒。
- 1030 无。

- 1031 无。
- 1033 无。
- 1034 实例间的交叉写：一个实例企图去修改一个 BLOCK，而另外一个实例想去获得此 BLOCK 的状态，这个实例产生最多一秒钟的等待。
- 1035 与 1034 一样，但是产生的原因是由于数据库处于热备的状态下。

Oracle 的后台进程 DBWR 获取可以写入的缓存并标记这些缓存为正在写入的状态，接着这些被收集的缓存中的数据将会被写入磁盘上的数据文件中，当所有的 I/O 完成后将清除在原来那些被标记的缓存上的标记，这个等待事件出现意味着 Oracle 想获取的 buffer 已经被标记为正在写入的状态，只有等标记被清除才能获取到相应的 buffer。在 Oracle 7.2 以前的版本中，只有当批处理中所有的 buffer 都被写入磁盘后标记才被清除，在这之后的版本，每个 buffer 写入磁盘后就将清除在这个 buffer 上的标记了。

增加更多的 Oracle 后台 DBWR 进程或者是采用异步 I/O 都将能减少这个等待事件的产生。

（20）free buffer waits 等待事件

这个等待事件出现的原因比较多，大致可以分为以下几种：

- 当一个数据文件从只读状态变成为可读写状态时，所有的 buffer gets 全部都被挂起了，就可能出现这个等待事件。已经存在的所有 buffer 都必须失效，因为它们没有链接到 lock elements（OPS/RAC 环境下时需要）。因此，只有当这些 buffers 失效完成后才能够被分配给数据库块地址。
- 当 Oracle 数据库需要从系统全局区（SGA）中读取一个 buffer 给一致性读（CR）操作、只读操作或者用于任何恢复模式中的操作时，也可能出现这个等待事件，此时可以加速 Oracle 后台的 DBWR 进程来获得较多的空闲 buffer。在检查了'free buffers inspected'之后也会出现这个等待事件，如果没有找到空闲的 buffer，Oracle 会等待 1 秒钟后继续试图去获取空闲的 buffer。

在 V$SESSION_WAIT 视图里面，这个等待事件有三个参数 P1、P2、P3。其中，P1 代表 Oracle 读取 buffer 而引起等待的数据文件的文件号，P2 代表数据文件中读取 buffer 的 BLOCK 的号，P3 代表要读取 buffer 的缓存中的 BLOCK（7.3.X 以上的版本）。

一般来说，这个等待事件都是由于 Oracle 的后台进程 DBWR 不能及时地将 buffer 写完到磁盘上的数据文件中而引起的，尽量将 I/O 平均分配到各个磁盘上，减少出现某个磁盘上 I/O 负载很高而引起 DBWR 进程写入慢的情况，可以通过操作系统上的 I/O 监控工具或者查询 V$FILESTAT 视图来获取相应的数据：

```
select name, phyrds, phywrts
from v$filestat a, v$datafile b
where a。file# = b。file#;
```

还可以通过查看数据文件上是否存在全表扫描来判断：

```
select name, phyrds, phyblkrd, phywrts
```

```
from v$filestat a, v$datafile b
where a。file# = b。file#
and phyrds!= phyblkrd;
```

需要注意的是，在应用中要避免漏建立索引，这样会引起 I/O 大幅度的增加，导致不必要的磁盘扫描，如果有多块硬盘来存储 Oracle 的数据文件，尽量使用操作系统的条带化软件来分布 Oracle 的数据文件，使得 I/O 分配均匀。此外，大量的磁盘排序会导致存在很多的脏缓存需要写完，因此，临时表空间中的数据文件最好能分配到不同的磁盘上，避免同一个磁盘上的 I/O 竞争。如果排序的 BLOCK 的检查点没有完成，将会存在于正常的缓存写批处理中，如果缓存写批处理中全部都被排序块给占满了，那其他的脏数据块就没法被写入导致前台的应用不得不等待分配空闲的 buffer。对于 Oracle 9i 之后的版本，因为排序使用的块通常都是来自临时表空间文件，不会进入到缓存中，因此，由于大量排序引起的这种等待在 9i 中基本上就不会存在了。

了解了在 Oracle 数据库 I/O 性能或者是响应时间低下时该如何去调整和优化数据库，还有一点很重要的事项需要提及——无论是何种情况，都应该先去检查操作系统上的日志文件，因为如果是本身在操作系统级别上出现了 I/O 问题，那么不管如何调整 Oracle 数据库都是徒劳的，所以必须首先保证在操作系统级别上 I/O 不存在问题，然后再去 Oracle 数据库中具体检查问题产生的原因。

不管用何种方法去解决 Oracle 数据库的 I/O 性能问题，关键都是先找出产生 I/O 性能问题的根本原因，然后想各种各样的办法去解决产生的原因就可以达到优化数据库的目的了。

7.2.3　如何检查 Oracle 数据库的性能

如何检查 Oracle 数据库性能和数据库性能情况，大体上管理员在日常运维中，需要重点关注的性能因素包含：检查数据库的等待事件，死锁检查及处理，CPU、I/O、内存性能，统计信息收集，检查缓冲区命中率。

1. 检查数据库的等待事件

常用的检查脚本如下：

```
set pages 80
set lines 120
col event for a40
select sid,event,p1,p2,p3,wait_time,seconds_in_wait from v$session_wait where
event not
like 'sql%' and event not like 'rdbms%';
```

如果数据库长时间持续出现大量 latch free、enqueue、buffer busy waits、db file sequential read、db file scattered read 等等待事件，就需要对其进行分析，查找可能存在问题的语句。下面列出定位问题 SQL 的方案。

（1）Disk Read 最高的 SQL 语句的获取

```
select sql_text from (select * from v$sqlarea order by disk_reads)
```

```
where rownum<=5 desc;
```

（2）查找前十条性能差的 SQL：

```
select * from (select parsing_user_id
executions,sorts,command_type,disk_reads,
sql text from v$sqlarea order by disk reads desc)
where rownum<10 ;
```

（3）等待时间最多的 5 个系统等待事件的获取：

```
select * from (select * from v$system event where event not like 'sql%' order by
total_waits desc) where rownum<=5;
```

（4）检查运行很久的 SQL：

```
column username format a12
column opname format a16
column progress format a8
select  username,sid,opname,round(sofar*100  /  totalwork,0)  ||  '%'  as
progress,time remaining,sql text  from  v$session longops  ,  v$sql  where
time_remaining <> 0 and sql_address=address and sql_hash_value = hash_value;
```

（5）检查消耗 CPU 最高的进程：

```
set line 240
set verify off
column sid format 999
column pid format 999
column s_# format 999
column username format a9 heading "ora user"
column program format a29
column sql       format a60
column osname format a9 heading "os user"
select p.pid pid,s.sid sid,p.spid spid,s.username username,s.osuser
osname,p.serial#
s_#,p.terminal,p.program program,p.background,s.status,rtrim(substr(a.sql_text,
1,
80)) sqlfrom v$process p, v$session s,v$sqlarea a where p.addr = s.paddr and
s.sql_address = a.address (+) and p.spid like '%&1%';
```

空间因素也是管理员需要重点关注的方面。随着时间推移，基于数据库的应用系统的广泛使用，产生的存储空间碎片会越来越多，将对数据库有以下两点主要影响。

（1）导致系统性能减弱。当要满足一个空间要求时，数据库将首先查找当前最大的自由范围，而“最大”自由范围逐渐变小，要找到一个足够大的自由范围已变得越来越困难，从而导致表空间中的速度障碍，使数据库的空间分配愈发远离理想状态。

（2）浪费大量的表空间。尽管有一部分自由范围（如表空间的 pctincrease 为非 0）将会被 SMON（系统监控）后台进程周期性地合并，但始终有一部分自由范围无法得以自动合并，浪费了大量的表空间。

- 检查碎片程度高的表：

```
SQL> select segment name table name,count(*) extents from dba segments where owner
not in ('sys', 'system') group by segment name having count(*)=(select max(count(*))
from dba_segments group by segment_name);
```

- 检查表空间的 I/O 比例：

```
SQL>select df.tablespace name name,df.file name "file",f.phyrds pyr, f.phyblkrd
pbr,f.phywrts pyw, f.phyblkwrt pbw from v$filestat f, dba_data_files df where
f.file# = df.file_id order by df.tablespace_name;
```

- 检查文件系统的 I/O 比例：

```
SQL>select
substr(a.file#,1,2)  "#",substr(a.name,1,30) "name",
a.status,a.bytes,b.phyrds,b.phywrts from v$datafile a, v$filestat b where a.file#
=
b.file#;
```

2. 死锁检查及处理

大家最早接触死锁这个概念可能是在操作系统课程中，说多个进程（线程）对一个可共享的资源进行请求时，可能出现死锁。死锁问题可以分为死锁检测、处理等多个子问题进行讨论。

其实，死锁问题绝不仅仅限制在操作系统乃至计算机科学领域。死锁存在两个必要条件，一个是多任务工作的并发，另一个是共享资源的独占性需求。只要一个系统（广义系统）中存在这两个前提，就认为可能出现死锁的情况。

死锁描述的是一种状态，当两个或两个以上的任务单元在执行过程中，因为请求资源出现等待，因资源永远不能获得而相互等待的状态。如果没有外力的作用，死锁状态会一直持续下去。死锁是伴随着多任务、并行操作产生的，在单任务情况下，一个任务单元可以使用并且独占所有资源，不存在资源等待的情况，所以没有死锁情况。在多任务系统环境下，多个任务之间存在资源共享和独占的需求，才可能出现死锁。

举一个简单的例子，现有任务 A、B 和资源 1、2，任务 A 独占了资源 1，任务 B 独占了资源 2，此时，任务 A 要资源 2，向任务 B 提出请求并等待，任务 B 要求资源 1，并且也等待。AB 两者均不释放所占有的资源，就造成了死锁。Oracle 是目前商业数据库市场上表现最优秀的并发数据库系统，同样存在死锁的威胁。存在并发、存在资源独占，就有锁或者类似锁概念的机制。Oracle 提供了多种类型锁和多种锁的机制，包括共享锁和独占锁，在进行各类型操作时，会自动对对象进行加锁解锁以及锁升级操作，最大可能地保证数据完整性。

如果出现死锁，Oracle 会如何处理呢？Oracle 的锁机制是建立在行锁一级的，在插入、更新行一级信息时，会加入独占锁内容。下面我们尝试模拟一下出现死锁的状态。两个 session 分别更新两条记录，在一个事务里再尝试更新对方记录，就可以引发死锁。针对死锁，可以通过如下实验帮助理解其产生的原因和场景。创建 t 表，准备测试环境。

【示例 7-2】创建 t 表

```
QL> desc t;
```

```
Name Type         Nullable Default Comments
---- ------------ -------- ------- --------
ID   NUMBER
COMM VARCHAR2(10) Y

SQL> select * from t where rownum<3;

        ID COMM
---------- ----------
         1 Tst1
         2 Tst2
```

模拟两个 session，分别针对 id=1、2 两条记录做文章。

```
//session1
SQL> select sid from v$mystat where rownum<2;

       SID
----------
       152

SQL> update t set comm='Tst1' where id=1;

1 row updated

//session2
SQL> select sid from v$mystat where rownum<2;

       SID
----------
       150

SQL> update t set comm='Tst2' where id=2;

1 row updated
```

在数据库里面查看锁的状态：

```
SQL> select * from v$lock where sid in (150, 152);

ADDR     KADDR           SID TYPE        ID1        ID2      LMODE    REQUEST
-------- -------- ---------- ---- ---------- ---------- ---------- ----------
333415A4 333415BC        152  TM       54599          0          3          0
33341668 33341680        150  TM       54599          0          3          0
3338A42C 3338A548        150  TX      393251        795          6          0
333B8954 333B8A70        152  TX      327686        779          6          0
```

此时，在两个 session 中，分别使用了行锁，独占（LMODE=6）锁住了两行。下面继续相互请求。在 session1 中，尝试更新 id=2 的记录：

```
SQL> update t set comm='tst2' where id=2;
```

session1（SID=152）被 hange 住，查看数据库锁状态：

```
SQL> select * from v$lock where sid in (150，152);

ADDR     KADDR           SID TYPE        ID1        ID2      LMODE    REQUEST
-------- -------- ---------- ---- ---------- ---------- ---------- ----------
33834398 338343AC        152 TX       393251        795          0          6
333415A4 333415BC        152 TM        54599          0          3          0
33341668 33341680        150 TM        54599          0          3          0
3338A42C 3338A548        150 TX       393251        795          6          0
333B8954 333B8A70        152 TX       327686        779          6          0
```



在 Oracle 中，hange 表示 session 正在等待资源被释放，表现就是停住操作，不断轮询资源。

session2 请求更新：

```
//session2 (SID=150)
SQL> update t set comm='tst1' where id=1;
```



此时，应当出现死锁状态。在系统中，出现了一瞬间的两个 session 互相 hange 的状态。

此时，session2 继续被 hange 住，原先的 session1 退出，状态如下：

```
//session1 (SID=152)
SQL> update t set comm='tst2' where id=2;

update t set comm='Tst2' where id=2
ORA-00060: 等待资源时检测到死锁
```

结果是 Oracle 在两个出现死锁的 session 中，随机寻找了一个 session，驳回了其被 hange 住的请求，维持另一方 session 的 hange 状态。



Oracle 此处的处理：只是驳回了一方的请求，并没有回滚该请求，也没有将另一方的 hange 状态解除。

显然，在这个过程中是 Oracle 内部的防护机制起了作用，防止了系统中死锁的发生。在 Oracle 中，存在某种轮询的机制，随时检查系统中出现的多会话被 hange 住的情况，一旦发生，就将一个 session 被 hange 住的请求退回，抛出 00060 错误。

在两个 session 互锁的情况下，Oracle 死锁检测程序可以起作用。当死锁结构复杂时，Oracle 的检测机制是不是会失效呢？使用 4~5 个 session 进行检查，虽然检测起效的时间有长有短，但最后都是将死锁的状态加以解除。

有一点需要注意，Oracle 解决死锁的方式只是将请求拒绝，并不是将事务回滚。所以，在解决死锁之后，其他被 hange 住的 session 依然还是会被 hange 住。所以，从应用程序和 PL/SQL

的角度来看，如果接收到了 00060 错误，那么应该做的工作就是回滚当前事务，解决整体的资源争用现象。

那么，Oracle 中死锁发生的概率是不是很高呢？答案是否定的。这个是由 Oracle 锁的特性所决定的。

（1）Oracle 对查询不加锁。Oracle 本身支持多版本一致读，如果当前的数据块正在被修改（独占）而并未提交，Oracle 会根据 SCN 查找日志和 Undo 空间，找到合适 SCN 的版本返回结果。所以，在查询时，是不需要加锁的。

（2）Oracle 数据操作使用行级锁（本质上是事务锁），实现最小粒度的独占范围。Oracle 在 DML 时，只会对操作的行进行独占锁定，而不是过大的数据单元（如页 page）。这样，就保证了不会引起过多的独占资源。

同时，Oracle 本身也提供了死锁监视程序功能，能及时发现死锁状态，并自动进行解锁。在这些机制下，Oracle 认为死锁发生的概率很低（起码自身不会引起死锁）。只有一种情况会引发死锁，就是开发人员手工提高加锁的级别。

在 Oracle 对于 SQL 的拓展中，有一部分是显示对象加锁。从 lock table XXX 到 select XXX for update。这些操作都会引起对对象锁级别的提升，这些都可能引发死锁的发生。

在前文中提到，死锁问题解决方法分为死锁监控和死锁处理。所谓死锁监控，就是发现死锁。在 Oracle 中，通过锁之间的连接关系，有自动的监控死锁、解决死锁的机制。但是实际中，特别是开发测试过程中，还是可能出现死锁的情况。下面列出一些网络上推荐的脚本，可以用于发现和解决死锁。

（1）发现锁对象和对应的 SID 发现锁定

```
select s.username, l.object_id, l.session_id, s.serial#, l.oracle_username,
l.os_user_name, l.process from v$locked_object l,v$session s where
l.session_id=s.sid;
```

（2）解决方法——kill 掉指定的 session

```
alter system kill session 'sid,serial#';  (其中 sid=l, session_id)
```

kill 掉 session，也可以在 EM 中进行，使用图形化界面。

（3）解决方法——kill 掉指定的 process

通过 sid 找到对应 service 进程的 id 信息，通过操作系统级别进行 kill。

```
select pro、spid from v$session ses,v$process pro where ses、sid=XX and ses。paddr=pro。
addr;
```

其中，sid 用死锁的 sid 替换。

```
exit
ps -ef|grep spid
```

其中，spid 是这个进程的进程号，kill 掉这个 Oracle 进程。

在 OS 层面 kill 进程的情况很多，比如使用 dbms_job 调用的 session，大部分时候都需要在 OS 层面 kill 掉。

一般情况下，应用程序对数据库的操作基本上是瞬时的（除非需要 Tunning）。只有涉及批量事务操作时，才会出现长时间对一个对象的独占。一旦这种情况出现，所独占的对象又是为其他会话（批量操作）所请求，就有可能出现死锁。

长时间的批量事务一般都组织在系统专门的作业管理器中，根据特定的调度规则进行运行。为了提高效率，作业管理器通常使用多线程技术，同时多个线程执行多个任务，这种情况下，出现死锁的概率也会大大提升。在这个问题上，考虑如下观点：

（1）按照模块划分线程，模块内作业顺序执行，模块间并行。一般来说，一个模块的作业之间会出现比较严重的资源请求共享，比如对同一个表中同一批数据的操作。这样很容易出现死锁状态。无论是手工解锁还是借助 Oracle 自身机制解锁，都至少引起一个作业的回滚。所以，根据模块进行划分，相同模块的作业尽可能顺序执行，减少表数据之间共用。对于不同模块之间的作业，操作更新的表范围差别比较大，所以不同业务模块之间作业进行并行。

（2）作业中显示进行错误 00060 的处理。在作业的开发中，要加入对于 00060 死锁错误的处理。如果接收到该错误异常，就要主动从应用层面进行操作回滚，接触其他作业阻塞情况。

（3）借助 Oracle 自身锁机制，监控长时间锁住对象和会话状况，尽快通知 DBA 手工解决死锁。

综上所述，死锁在 Oracle 中是一种比较少见的情况，而且自身有相应的监控防护机制。作为应用开发者的我们，也要尽可能地合理化应用处理结构，尽可能不要显示的锁定对象和设计不合理的并行操作，加大出现死锁的概率。

- 查询目前锁对象信息:

```
col sid for 999999
col username for a10
col schemaname for a10
col osuser for a16
col machine for a16
col terminal for a20
col owner for a10
col object name for a30
col object type for a10
select sid,serial#,username,schemaname,osuser,machine,
terminal,program,owner,object_name,object_type,o.object_id
from dba_objects o,v$locked_object l,v$session s
where o.object_id=l.object_id and s.sid=l.session_id;
```

- Oracle 级 kill 掉该 session:

```
alter system kill session '&sid,&serial#';
```

- 操作系统级 kill 掉 session:

```
#>kill -9 pid
```

3. 检查 CPU、I/O

检查数据库 CPU、I/O、内存性能，记录数据库的 CPU 使用、IO、内存等使用情况，可以使用操作系统自带的性能监控工具 vmstat、iostat、sar、top 等命令进行信息收集并检查这些信息，判断资源使用情况、CPU 使用情况：

```
 [root@sale8 ~]# top
top - 10:29:35 up 73 days, 19:54, 1 user, load average: 0.37, 0.38, 0.29
Tasks: 353 total,  2 running, 351 sleeping,  0 stopped,  0 zombie

Cpu(s):  1.2% us,  0.1% sy,  0.0% ni, 98.8% id,

0.0% wa,  0.0% hi,  0.0% si

Mem: 16404472k total, 12887428k used,  3517044k free,   60796k buffers
Swap: 8385920k total,  665576k used,  7720344k free, 10358384k cached
```

4. 统计信息收集

定期做统计分析对于采用 Oracle Cost-Based-Optimizer 的系统，需要定期对数据对象的统计信息进行采集更新，使优化器可以根据准备的信息做出正确的 explain plan。

在以下情况下更需要进行统计信息的更新：

- 应用发生变化。
- 大规模数据迁移、历史数据迁出、其他数据的导入等。
- 数据量发生变化。

查看表或索引的统计信息是否需更新，例如：

```
SQL>select table name,num rows,last analyzed from user tables where
table name ='dj nsrxx'
SQL>select count(*) from dj_nsrxx
```

例如 num_rows 和 count(*)，如果行数相差很多，就需要更新统计信息，建议一周做一次统计信息收集：

```
SQL>exec sys.dbms stats.gather schema stats(ownname=>'CTAIS2', cascade=>
TRUE, degree => 4);
```

5. 检查缓冲区命中率

可以通过下面的脚本直接查看数据库的缓冲区命中率：

```
SQL> select a.value + b.value logical_reads,
c.value phys_reads,
round(100*(1-c.value/(a.value+b.value)),4) hit ratio
from v$sysstat a,v$sysstat b,v$sysstat c
where a.name='db block gets'
and b.name='consistent gets'
and c.name='physical reads' ;
LOGICAL_READS PHYS_READS  HIT_RATIO
```

```
------------- ---------- ----------
1273645705   71191430    94.4104
```

如果命中率低于 90%，就需要加大数据库参数 db_cache_size 检查共享池命中率：

```
SQL> select sum(pinhits)/sum(pins)*100 from v$librarycache;
SUM(PINHITS)/SUM(PINS)*100
--------------------------
99.5294474716798
```

如果低于 95%，就需要调整应用程序，使用绑定变量，或者调整数据库参数 sharedpool 的大小。

（1）检查排序区：

```
SQL> select name,value from v$sysstat where name like '%sort%';

NAME
---------------------------------------------------------------- ----------

VALUE

sorts (memory)                                      6135534
sorts (disk)                                          8
sorts (rows)                                          2264742084
```

如果 disk/(memoty+row) 的比例过高，就需要调整 sort_area_size(workarea_size_policy=false 或 pga_aggregate_target(workarea_size_policy=true)。

（2）检查日志缓冲区：

```
SQL> select name,value from v$sysstat where name in ('redo entries','redo buffer
allocation
retries');

NAME
---------------------------------------------------------------- ----------
redo entries
redo buffer allocation retries

VALUE
27663705
880
```

如果 redo buffer allocation retries/redo entries 超过 1%，就需要增大 log_buffer。

7.2.4　问题处理方法与思路

性能调优主要有主动调优和被动调优，主动调优在前面已经进行了阐述，被动调优主要有以下方法。

- 确定合理的性能优化目标测试并记录当前的性能指标。
- 确定当前存在的 Oracle 性能瓶颈（Oracle 中何处存在等待，哪个 SQL 语句与此有关）。
- 确定当前的操作系统瓶颈优化相关的组件（应用、数据库、I/O、连接 OS 及其他）跟踪，实施变化管理制度测试并记录目前的性能指标重复直至达到既定的优化目标。不要对并非性能瓶颈的部分进行优化，否则可能引起额外的问题。正如任何聪明的人会告诉你的："如果还未坏，千万不要修"。更重要的是，一旦既定的优化目标已经达到，就务必停止所有的优化。
- 获取 Oracle 的性能指标（测试前及测试后）必须在峰值处理时测试并获取系统在优化前和优化后的性能指标。
- 数据采集不应在数据库 instance 刚刚起动后进行。
- 测试数据应在峰值期间每过 15 分钟进行一次。
- 初始化参数 TIMED_STATISTICS 应该被设为 TRUE。

通过运行以下脚本开始快照：

```
$ORACLE_HOME/rdbms/admin/utlbstat.sql.
```

通过运行以下脚本结束快照：

```
$ORACLE_HOME/rdbms/admin/utlestat.sql.
```

完成 utlestat.sql 操作后，会在当前目录中生成名为"report.txt"的文件，包含系统的性能数据。该报告包括每 15 分钟捕获的所有与 Oracle 例程相关的参数。

7.3 寻找问题根源

首先通过查看 v$system_event 事件开始系统事件的问题诊断。接着查看 v$session_event，找出引起或经历等待事件的进程。然后通过 v$session_wait 获得事件的细节。同时，应该进一步通过 OS 进行深入分析，了解核心的 CPU、内存和 IO 状态参数。最后，结合两种不同的诊断结论，找出系统瓶颈所在。

7.3.1 System_Event 事件

v$system_event 可以从全局的角度查看 Oracle 系统中的所有事件。尽管它并不包括任何进程级的信息（当前或历史），但却可以显示上次例程弹出后总的等待时间。这种动态性能视图中的数据，会在下次例程起动时清零。出于这种原因，这种视图中的数据应该在不同时段进行抽样。

7.3.2　Session_Event 事件

v$session_event 视图在进程级提供与 v$system_event 相同的信息（SID 等）。这种视图可以从“system-wideevents”级进一步钻取，到达进程级，以确定哪个进程引起或经历了等待事件。

7.3.3　Session_Wait

v$session_wait 视图在特定事件的进程级提供低层次的信息挖掘。不同于其他视图，这种方式可以“实时”获取进程级的等待信息。这是真正有用的信息。切记，每次查看这一视图得到的结果可能不一样。这可能与数据库中当前的活动有关。

7.3.4　应用优化

从统计（和现实）的角度看，80%的 Oracle 系统性能问题可以通过 SQL 代码优化来解决。任何应用优化的过程，不外乎是索引优化、全表扫描、并行机制改进和选择正确数据组合方法的过程。这正是要达到最佳应用性能所必须考虑的因素。没有 SQL 的优化，就无法实现高性能的应用。良好的 SQL 语句可以减少 CPU 资源的消耗，提高响应速度。同时，优化后的 SQL 语句还可以提高应用的可扩展性，这是除增加大量内存外，任何其他硬件手段也无法实现的。

7.3.5　内存调优

需要配置的主要初始化参数，以下是一些已知与例程优化关系最密切的核心 Oracle 初始化参数。它们都会影响 Oracle 及 SGA 区的活动。任何对这些参数的改动，在实施到生产环境之前都必须进行测试。一旦改变了生产环境的参数，就必须对相关的 Oracle 动态性能指标和操作系统的性能进行监测，寻找可能由此产生的异常现象。

（1）DB_BLOCK_SIZE，该参数在数据库建立前设定，决定了数据库中每个数据块的大小。只有重新建立数据库，才有可能改变该参数。db_block_size 的配置应遵循以下公式：

DB_BLOCK_SIZE=FILE SYSTEM BLOCK SIZE>=O-SPAGE SIZE

这可以确保 Oracle 获得最佳 I/O 性能，同时不会由于冗余或不必要的 I/O 给 I/O 子系统带来压力。

（2）DB_BLOCK_BUFFERS，该参数决定了 SGA 区数据库缓冲区中的块数量。由于这是 Oracle 读取和写入的区域，它的不正确配置会引起严重的 I/O 性能问题。尽管缓冲区的大小与应用性质、数据库大小、同步用户数等无关，但它的确是 SGA 区中最大的组件。经常可以看到缓冲区占用 75%~80%SGA 区内存的情况。另外，这一参数设置过大，也会引起整个系统的内存不足，引起操作系统过多的读写操作。该参数及 SHARED_POOL_SIZE 通常是两个最重要的 SGA 优化目标。只有当数据库缓冲率长时间低于 70%时，才需要增加其大小。即使在这种情况下，也需要进一步审查应用的性能和整个系统的吞吐性。若存在延迟性的应用设计问

题，则无论数据库缓冲区的大小如何，缓冲和读写率都不会有太大改变。在实际调优中，也曾发现由于 SQL 语句的问题而出现缓冲率很高但仍存在全系统性能问题的情况。

（3）SHARED_POOL_SIZE，该参数按字节数设定，定义了 SGA 中共享区的大小。该组件的大小严重依赖于应用的类型（该应用是重用 SQL 还是生成动态 SQL，等等）。同时它也取决于同步用户的数量，以及实例是否被配置成支持多线程服务器（MTS）。如果该应用采用了 MTS 配置，则共享区应该明显增加，因为游标状态和用户进程数据等程序全局区域（PGA）都被置入了共享区。有关多数应用的 SHARED_POOL_SIZE 大小设置，可以从每 10 个同步用户 16MB 共享区开始。这不是一成不变的，因为应用的性质最终会决定该组件的大小。只有当库缓冲和字典缓冲使用率一直低于 90%时，才需要关注这一参数。如果应用并未采用绑定变量和自适应共享游标时，内存的数量并不会使缓冲使用率高于 90%。共享区过大会导致处理时间增加，甚至 SQL 语句的挂起。如果应用不能有效地重用 SQL，则无论配置多大的库缓冲或字典缓冲都无济于事，不能改善缓冲使用率。另一个值得考虑的因素是需要随时使用的存储 PL/SQL 代码数量。应用的核心包可以通过查看 DBA_SOURCE、USER_SOURCE 得以确认，其大小通过查询 DBA_OBJECT_SIZE 了解。另外，为了确定存储 PL/SQL 是否被置于内存，可以查询动态性能视图 V$DB_OBJECT_SIZE。同时，包 DBMS_SHARED_POOL 中的程序大小可被用于确定应用中大包的规模。

（4）LOG_BUFFER，根据字节设定，该参数定义了 SGA 缓冲区中 redo log 的大小。默认值通常是数据库块大小的四倍，这对于多数环境并不是最佳的。对于中型的 Oracle 环境，其结构应该为 512KB 左右。对该存储结构而言，更大并不意味着更好。超过 1MB 就可能有问题。需要监控 V$SESSION_WAIT 中 log buffer space 的等待事件，以优化该内存结构。需要提醒的是，在线 redo log 文件的大小设置不当会引起 redo 请求的等待。

（5）DB_WRITERS，该参数可以针对所有文件系统支持，且不可使用 DirectI-O 的 Oracle 实施设定。这并不需要与 raw partitions 一起使用，因为异步 I-O 更佳。建议将该参数设定为（2*独立磁盘驱动器数量/卷）。该参数只有在 report.txt 中的“average write queue length”持续高于 1 时，才需要设定。在 Oracle 8.0 和更高版本中，该参数已不再被支持，而被其他两个名为 DB_WRITER_PROCESSES 和 DBWR_IO_SLAVES 的参数取代。若需要设置 DB_WRITER_PROCESSES 值高于 8，则 DB_WRITER_PROCESSES 可被设为 1，且 DBWR_IO_SLAVES 可被设为“n”，其中 n 的值必须设置为（2 *独立磁盘驱动器数量/卷）。

7.3.6 I/O 优化

I/O 优化是系统优化中的一个关键步骤，还涉及其他任务，将文件在不同驱动器/卷中进行分布，采用优化分区技术、确定 I/O 子系统瓶颈、确定控制器瓶颈并根据应用的类型选择最佳的 RAID 级。I/O 优化应该在全面了解 Oracle 及 Oracle RDBMS 结构之后进行。应该在进行 I/O 优化前后实施 I/O 数据监控，如平均服务时间、IOPS、平均磁盘队列长度等。

7.3.7　竞争优化

多数与 Oracle 有关的竞争问题可以通过主动配置管理相关的初始化参数进行。不恰当地配置 init.ora 中的锁参数可能引起竞争。为了不打破其中的平衡，所需的参数可进行配置并主动得以处理。包括表在内的数据库对象可能存在两个竞争点。第一个是所配置的“freelists”的数量（默认值为 1）。freelist 结构维护着表中可用于插入的块。对于存在大量同步插入的表，有必要配置该结构。为了以主动方式处理 freelist 竞争，必须在建立表时配置 FREELISTS。可考虑的最佳值为(2*CPU 数量)。V$WAITSTAT 不可能指示存在 freelist 竞争，除非存在 freelist 组，而这种设置只存在于 Oracle Parallel Server 中。即便如此，也无法了解哪个表存在竞争中。主动式的 freelist 竞争调优可以事先预防问题出现。资源竞争的第二个来源与索引有关，即对象块头中配置的事务槽数量。事务槽是块头中的区域，是事务处理进程采用自身识别号进行注册，以便任何被修改的更能够通过特定事务槽数量在低层得以识别的地方。如果所有现存的事务槽已经被其他事务占用，服务器进程就会从块的 PCTFREE 中请求 23 个字节，建立一个新的槽。这种情况适用于存在大量同步事务的对象。对于事务槽的竞争，需要设置 INITRANS 参数。对于块大小为 8KB 的数据库，多数情况下，4 为最佳设置，占用的空间仅为 92 字节，却可以大大减少运行时的故障和性能问题。

7.4　小结

Oracle Database 运作速度优化与主机配置，并无绝对的标准！除了靠 DBA 平日对 Oracle Instance 运作速度观察、用户反馈之外，Oracle 提供了一些数字指针（如 Cache 命中率）与建议方案供 DBA 作为优化依据。另外，配合实体文件位置规划、Row chaining 问题处理、SQL 分析与优化，可以大幅降低 Oracle Instance 所造成的运作速度低下与异常情况发生的概率。优化方面的经验，有少部分技巧可以通过书本获取，但大部分的优化技巧需要从日常的数据库管理工作中不断地积累，希望读者可以在真实的生产环境中多动手、多思考，总结适合自己的优化方式或者最佳实践。

数据库的日常管理，无论是基础管理还是性能管理，都需要使用工具连接数据库进行相应的操作，在下一章节，将向读者介绍 Oracle 数据库最基本的管理工具。

第 8 章

Oracle常用管理工具

Oracle 的 SQL*Plus 是与 Oracle 进行交互的客户端工具。在 SQL*Plus 中，可以运行 SQL*Plus 命令与 SQL*Plus 语句。SQL*Plus 工具是随 Oracle 数据库服务器或客户端的安装而自动进行安装的管理与开发工具，Oracle 数据库中所有的管理操作都可以通过 SQL*Plus 工具完成，同时开发人员利用 SQL*Plus 可以测试、运行 SQL 语句和 PL/SQL 程序。

8.1 SQL*Plus 工具及其使用

通常所说的 DML、DDL、DCL 语句都是 SQL*Plus 语句，它们执行完后，都可以保存在一个被称为 sql buffer 的内存区域中，并且只能保存一条最近执行的 SQL 语句，可以对保存在 sql buffer 中的 SQL 语句进行修改，然后再次执行，SQL*Plus 一般都与数据库打交道。SQL*Plus 是一个命令行界面的查询工具，拥有自己的命令和环境。

除了 SQL*Plus 语句，在 SQL*Plus 中执行的其他语句称为 SQL*Plus 命令。它们执行完后，不保存在 sql buffer 的内存区域中，一般用来对输出的结果进行格式化显示，以便于制作报表。下面将对该工具的使用方法和常用命令进行介绍。

8.1.1 启动 SQL*Plus

启动 SQL*Plus 可以在命令行上直接输入 sqlplus，从 Oracle 程序组的“应用程序开发”中选择 SQL Plus，启动语法格式为：

```
sqlplus [ [<option>] [logon] [<start>] ]
```

其中，option 部分的主要选项为：

- Help：显示 SQL*Plus 程序的使用帮助信息。
- Version：显示 SQL*Plus 版本号。
- Silent：要求以哑模式（静默的模式，在这种模式下，程序不会与用户进行交互）启

动和运行 SQL*Plus。

logon 参数指出登录相关信息，其格式为：

```
{ [用户名[/口令] | /][@连接标识符] } [AS {SYSOPER |SYSDBA}] | /NOLOG
```

start 参数指出 SQL*Plus 启动后立即执行的脚本文件名称及其参数，其语法格式为：

```
@{url 地址|脚本文件名[.扩展名]} [参数 ...]
```

语法结构看上去稍微复杂一些，实际操作中使用 SQL*Plus 登录数据库使用较多的方法如图 8.1 所示，使用 sysdba 身份登录数据库。这种登录方式也可以称为本地认证登录，因为用户已经登录数据库的操作系统，Oracle 认为用户本身是具备管理数据库的最高权限的。

```
$ sqlplus / as sysdba

SQL*Plus: Release 12.2.0.1.0 Production on Mon Apr 16 11:23:59 2018

Copyright (c) 1982, 2016, Oracle.  All rights reserved.

Connected to:
Oracle Database 12c Enterprise Edition Release 12.2.0.1.0 - 64bit Production
```

图 8-1 sqlplus 登录数据库

或者也可以在登录时只输入 sqlplus 命令，程序会提示用户输入用户名和密码，该方法使用户可以自主选择使用哪个数据库用户登录。

8.1.2 关闭 SQL*Plus

关闭 SQL*Plus 可以采用以下两种方法：

- 异常关闭：直接关闭 SQL*Plus 窗口，或者由于其他原因导致 SQL*Plus 与 Oracle 数据库服务器之间的连接异常中断。未完成的事务被 Oracle 实例回滚。
- 正常关闭：执行 SQL*Plus 命令 exit 关闭。未完成事务的结束方式由 SQL*Plus 环境参数 EXITCOMMIT 的值决定：值为 ON（默认值），提交事务；值为 OFF，回滚。

8.1.3 设置变量

在 SQL*Plus 下可以定义变量、设置参数，参数改变当前会话的环境设置等。变量分为 SQL*Plus 预定义变量和用户变量。定义 SQL*Plus 变量 define 命令查看预定义变量。

例如，启动 SQL*Plus 以 scott 用户连接数据库，查看预定义变量：

```
SCOTT@orcl SQL >define
```

define 命令定义用户变量，语法：

```
define variable_name ='value'
```

例如：

```
SCOTT@orcl SQL >define dept_no = 10
```

查看变量：

```
SCOTT@orcl SQL >define dept no
DEFINE DEPT_NO        = "10" (CHAR)
```

使用 undefine 命令释放变量，例如释放 dept_no 变量：

```
SCOTT@orcl SQL >undefine dept_no
```

8.1.4 设置 SQL*Plus 环境参数

SQL*Plus 参数也称作系统变量，由 set 命令设置，用于改变 SQL*Plus 当前会话的环境设置。常用的参数如表 8-1 所示。

表 8-1　SQL*Plus 参数表

参数	说明
SQLPROMPT	设置 SQL*Plus 命令提示符的格式，其默认值为“SQL>”
LINESIZE	设置输出中一行上可以显示的字符总数，默认为 80
PAGESIZE	设置输出中每页显示的行数，默认为 14
TIME	是否显示当前时间，取值为 ON 或 OFF，默认为 OFF
TIMING	是否显示每条 SQL 语句或 PL/SQL 块运行时间统计信息，取值为 ON 或 OFF，默认为 OFF
AUTOCOMMIT	设置 SQL 或 PL/SQL 语句执行后是否自动提交，其值为 ON、OFF 或 n，分别表示为执行每条语句后自动提交、不自动提交、执行 n 条语句后自动提交一次，默认为 OFF
EXITCOMMIT	指出 SQL*Plus 下执行 EXIT 命令时对未提交事务的默认操作是 COMMIT 还是 ROLLBACK，对应的取值分别为 ON 和 OFF，默认为 O
SERVEROUTPUT	指出在 SQL*Plus 内是否显示存储过程或 PL/SQL 块的输出信息，这些输出信息由 DBMS_OUTPUT.PUT_LINE 产生，默认为 OFF
ARRAYSIZE	从数据库中提取的行数，默认为 15
COLSEP	选定列之间的分隔符号，默认为空格
FEEDBACK	显示反馈行信息的最低行数，默认为 6
HEADING	是否显示列标题，默认为 ON
AUTOTRACE	是否为成功执行的 DML 语句产生一个执行报告

例如，修改 SQL*Plus 默认提示符：

```
SQL> set sqlprompt " user'@' connect identifier > "
SCOTT@orcl_dbs >
```

使用不同的连接字符串重新连接：

```
SCOTT@orcl_dbs > conn scott/tiger@dbs:1525/orcl.jmu.edu.cn
```

已连接：

```
SCOTT@dbs:1525/orcl.jmu.edu.cn >
```

显示环境参数：

```
SQL> show all
SQL> show variable_name
```

8.1.5　设置 SQL*Plus 配置文件

DBA 或用户可以使用配置文件设置 SQL*Plus 环境，这样每次建立连接后自动运行配置文件，即可得到相同的环境设置。SQL*Plus 配置文件包括两类：

- 站点配置文件由 DBA 在 Oracle 数据库服务器上建立，文件名为 glogin.sql，存储在 ORACLE_HOME 下的 sqlplus\admin\子目录内。影响连接该数据库（站点）的所有用户。
- 用户配置文件由用户在客户端创建，文件名为 login.sql，存储于当前目录或者 SQLPATH 注册项所指定的目录（Windows 下为%ORACLE_HOME%\dbs）内。只影响当前用户。



如果站点配置文件和用户配置文件对同一个参数进行设置，那么由于用户配置文件执行在后，因此其设置将覆盖站点配置文件中的设置。

例如，在 glogin.sql 内添加以下设置，使每个用户在连接后修改 SQL 提示符和输出的行、页数据长度。

```
set sqlprompt " user'@' connect identifier  date> "
set linesize 120
set pagesize 24
```

8.1.6　编辑执行 SQL 语句

在 SQL*Plus 中可以编辑执行的内容分为 3 类：

- SQL*Plus 命令；
- SQL 语句；
- PL/SQL 语句块。

用户需要结束编辑状态，可以执行以下任意操作：

- 在新行的开始直接输入句点(.)：只结束编辑状态，而不执行 SQL 语句。
- 在空行上直接按回车键：结束编辑，但不执行 SQL 语句。
- 输入分号(;)：结束编辑状态并执行已输入 SQL 语句。

- 在一行上输入斜杠(/)：结束编辑并执行已输入的 SQL 语句。

8.1.7 编辑执行 SQL*Plus 命令

输入 SQL*Plus 命令后按回车键，即可结束编辑状态，并立即执行。在具体执行时，用户输入 SQL*Plus 命令时，可以使用缩写。例如，连接命令 CONNECT，简写是 CONN，作用是进行用户切换或连接到新的数据库。语法如下：

```
CONNECT [username]/[password][@hoststring]
```

DISCONNECT 是断开与数据库的连接。



DISC 命令的作用仅仅是断开与数据库的连接，不退出 SQL*Plus 环境！

日常其他 sqlplus 命令举例：

- APPEND——将指定的文本追加到缓冲区内当前行的末尾。
- CHANGE——修改缓冲区中当前行的文本。
- DEL——删除缓冲区中当前行的文本。
- N——用数值定位缓冲区中的当前行。
- INPUT——在缓冲区当前行的后面新增加一行文本。
- EDIT——以文本编辑器方式打开缓冲区，进行编辑。



使用 EDIT 命令时，缓冲区中必须存在信息。

【示例 8-1】在 SQL*PLUS 中编辑 SQL 缓冲区中的 SQL 语句

```
SQL> select deptno,dname from dept;
显示结果:
   DEPTNO DNAME
--------- --------------
       10 ACCOUNTING
       20 RESEARCH
       30 SALES
       40 OPERATIONS

SQL> L 1
  1* deptno,dname
SQL>A,loc
  1* deptno,dname,loc

SQL> L
  1  select deptno,dname,loc  from dept
SQL>/
```

显示结果：

```
DEPTNO DNAME         LOC
--------- -------------- -------------
      10 accounting    new york
      20 research      dallas
      30 sales         chicago
      40 operations    boston
```

8.2 常用的 SQL*Plus 命令

建议读者跟随以下示例内容进行练习，掌握比较常用的 SQL*Plus 工具的使用方法。

8.2.1　连接数据库

（1）执行一个 SQL 脚本文件，首先通过 SQL*Plus 连接到数据库。

【示例 8-2】连接到数据库

```
sqlplus / as sysdba
SQL>start file name
SQL>@file_name
```

（2）重新运行上一次运行的 SQL 语句。

```
SQL>/
```

（3）将显示的内容输出到指定文件。

```
SQL> SPOOL file_name
```

在屏幕上的所有内容都包含在该文件中，包括输入的 SQL 语句。

（4）关闭 spool 输出。

```
SQL> spool off
```

只有关闭 spool 输出，才会在输出文件中看到输出的内容。

（5）显示一个表的结构。

```
SQL> desc table_name
```

8.2.2　格式化命令

SQL*Plus 虽然是 DBA 最为经常使用的 Oracle 客户端工具，但是它在输出结果格式化上不是很好，如折行、分页不好等，所以一般启动 sqlplus 后多少都要做些设置，如 col、linesize、pagesize 等。下面着重介绍 col 命令的使用方法。

col 命令是主要格式化列的显示形式，该命令有许多选项，命令的格式如下：

```
COL[UMN] [{ column|expr} [ option ...]]
```

OPTION 选项可以是如下子句：

```
ALI[AS] alias
CLE[AR]
FOLD_A[FTER]
FOLD_B[EFORE]
FOR[MAT] format
HEA[DING] text
JUS[TIFY] {L[EFT]|C[ENTER]|C[ENTRE]|R[IGHT]}
LIKE { expr|alias}
NEWL[INE]
NEW_V[ALUE] variable
NOPRI[NT]|PRI[NT]
NUL[L] text
OLD_V[ALUE] variable
ON|OFF
WRA[PPED]|WOR[D_WRAPPED]|TRU[NCATED]
```

接下来，通过具体的示例介绍 col 格式化命令的使用技巧。

【示例 8-3】改变默认的列标题

```
column column_name heading column_heading
For example:
SQL>select * from dept;
    DEPTNO DNAME                          LOC
---------- ---------------------------- ---------
        10 ACCOUNTING                    NEW YORK
SQL>col  loc heading location
SQL>select * from dept;
   deptno dname                          location
--------- ---------------------------- -----------
       10 ACCOUNTING                    NEW YORK
```

【示例 8-4】将列名 ENAME 改为新列名 EMPLOYEE NAME 并将新列名放在两行上

```
Sql>select * from emp
Department  name          Salary
---------- ---------- ----------
        10 aaa                11
SQL> column ename heading 'employee|name'
Sql>select * from emp
          Employee
Department  name          Salary
---------- ---------- ----------
        10 aaa                11
note: the col heading turn into two lines from one line.
```

【示例 8-5】改变列的显示长度

```
FOR[MAT] format
Sql>select empno,ename,job from emp;
     EMPNO ENAME      JOB
```

```
---------- ----------    ---------
      7369 SMITH      CLERK
      7499 ALLEN      SALESMAN
7521 WARD       SALESMAN
Sql> col ename format a40
     EMPNO ENAME                                 JOB
----------  ----------------------------------------       ---------
      7369 SMITH                                 CLERK
      7499 ALLEN                                 SALESMAN
      7521 WARD                                 SALESMAN
```

【示例 8-6】设置列标题的对齐方式

```
JUS[TIFY] {L[EFT]|C[ENTER]|C[ENTRE]|R[IGHT]}
SQL> col ename justify center
SQL> /
     EMPNO           ENAME                JOB
----------  ----------------------------------------      ---------
      7369 SMITH                                 CLERK
      7499 ALLEN                                 SALESMAN
7521 WARD                                   SALESMAN
```

对于 NUMBER 型的列，列标题默认在右边，其他类型的列标题默认在左边。

【示例 8-7】不让一个列显示在屏幕上

```
NOPRI[NT]|PRI[NT]
SQL> col job noprint
SQL> /
     EMPNO           ENAME
----------    ----------------------------------------
      7369 SMITH
      7499 ALLEN
7521 WARD
```

【示例 8-8】格式化 NUMBER 类型列的显示

```
SQL> column sal format $99，990
SQL> /
Employee
Department Name       Salary   Commission
---------- ---------- --------- ----------
30         ALLEN       $1，600   300
```

【示例 8-9】显示列值时，如果列值为 NULL 值，用 text 值代替 NULL 值

```
COMM NUL[L] text
SQL>COL COMM NUL[L] TEXT
```

【示例 8-10】显示列的当前显示属性值

```
SQL> column column_name
```

下面介绍几个关于格式化方面的其他命令与使用方法。

【示例 8-11】将所有列的显示属性设为默认值

```
SQL> clear columns
```

【示例 8-12】屏蔽掉一个列中显示的相同的值

```
BREAK ON break_column
SQL> break on deptno
SQL> select deptno, ename, sal from emp  where sal < 2500 order by deptno;
DEPTNO     ENAME        SAL
---------- ----------- ---------
10          CLARK       2450
MILLER     1300
20          SMITH       800
ADAMS      1100
```

【示例 8-13】列值变化时在值变化之前插入 n 个空行（在【示例 8-12】的结果显示中操作）

```
BREAK ON break_column SKIP n

SQL> break on deptno skip 1
SQL> /
DEPTNO ENAME SAL
---------- ----------- ---------
10 CLARK 2450
MILLER 1300

20 SMITH 800
ADAMS 1100
```

【示例 8-14】显示对 BREAK 的设置

```
SQL> break
```

【示例 8-15】删除设置的方法

```
SQL> clear breaks
```

8.2.3 SET 命令

在设置 NUMBER 数据的显示宽度、设置每页的行数、设置列的宽度等方面除了 col 命令之外还可以使用 SET 命令改变这些系统变量，可用 SHOW 命令列出具体方法。该命令包含许多子命令：

```

set colsep' '      --行的标题列的分隔符
set linesize(line) --设置 sqlplus 输出的最大行宽
set pagesize       --设置页面的最大行数，默认为 24，为了避免分页，可设定为 0
set serveroutput on|off
set echo on  --显示文件中的每条命令及其执行结果，默认为 on
set echo off --不显示文件中的命令，只显示其执行结果
```

```
set term on  --查询结果既显示于假脱机文件中(spool 指定输出的文件)，又在 SQLPLUS 中显示
set term off --查询结果仅仅显示于假脱机文件中(spool 指定输出的文件)
set heading off  --让结果行的标题不显示，默认为 on
set heading on   --让结果行的标题显示
set trimout on --去除标准输出每行的拖尾空格，默认为 off
set trimspool on --去除重定向（ spool）输出每行的拖尾空格，默认为 off
set timing off   --显示每条 sql 命令的耗时，默认为 off
set verify off   --是否显示替代变量被替代前后的语句
exit            --退出
```

根据以上内容，当用户需要执行一个.sql 文件时，可以在.sql 文件的开头使用 set 命令对输出文本的结构进行控制。

8.3 小结

希望读者可以通过本章的学习，了解 SQL*Plus 工具的使用方法和一些小技巧。其实，在软件市场中，Oracle 数据库管理工具的种类还有很多，包括 Oracle 自己的图形界面管理工具等，对数据库的日常运维都可以发挥非常大的作用。但是无论这些软件的功能多么强大，在网络连接不通的情况下，SQL*Plus 是 DBA 管理数据库、处理数据库故障的唯一选择。

经过以上章节的学习，Oracle 数据库的主要知识点已向读者做了介绍，虽然 Oracle 数据库的版本在不断地演化、推陈出新，Oracle 的基本功能点并没有发生变化，其体系结构或者功能在新的版本中只是有所加强或者创新。目前 Oracle 数据库的版本是一种基于“云”概念的数据库版本，在下一章节中将介绍 Oracle 12c 版本的一项重要功能特点，即多租户数据库功能。

第 9 章

Oracle 12c多租户功能

Oracle Database 12c 版本的发布迎合了目前云计算的发展潮流，是前些年 SAAS 的延伸，软件就是服务。以后使用软件，就像电源插座一样，无论到哪里，接上插头就可以使用。Oracle Database 本身是系统软件。系统软件不必是云，就好像 Linux 一样，暂时还无法想象 Linux 怎么变成云，但系统软件要能很好地支持上层的“云应用”。12c 中的 c 是指 Oracle 将更好地支持上层的“云应用”。

9.1 Oracle 12c 版本新特性

Oracle 12c 是甲骨文公司的重量级产品，有约 500 多项的更新。针对 12c 的主要变化可以从两方面来理解，一是功能，二是内部原理。

先说功能上的变化，12c 的新功能的确很多，最大的亮点就是 Pluggable Database，即 PDB，“可插拔数据库”。网上已经有很多相关的文章，每一个 PDB 都是一个单独的小数据库，若干个 PDB 组合在一个大的 CBD 中，共同构成一个大数据库。需要迁移数据时，可以将某个 PDB 拔出，插入另外的 CBD。这个特性将使未来的数据迁移、数据流动更加方便。

还有 ASM 方面，ASM 和数据库实例可以分别放在两台主机上。ASM 和数据库实例的分离至少说明 Oracle 以后有可能要为 ASM 添加更多的功能，如果 ASM 和数据库实例挤在同一机器，ASM 功能太多，占用资源太多，势必会影响数据库实例的运行。而且，分离之后，ASM 就有点像存储的控制器了。存储控制器通过网络把 LUN 输送给主机，ASM 通过网络，把 DiskGroup 输送给数据库。

值得一提的是，12c 中 Oracle 开始对 EM 做减法，推出了轻量级 EM，安装简单、占用资源少，问题也更少。使用轻量级 EM，将大大方便 DBA 的操作，但同时图形化、智能化工具的普及也必将提高 DBA 的要求。未来 DBA 必将向“一专多能”方向发展，但要注意，千万不能只“多能”。有“一专”为基础的“多能”才是如虎添翼。

另外，从内部原理、运行机制上说，Oracle 也有不小的变动。Oracle 从来没有停止精益求精的步伐。比如，12c 下 LGWR 已经可以有多个，多 LGWR 无疑将大大提高并发事务量。为达到这个目的，Oracle 必然对 Redo 这块机制有很大的变动，Redo 这块的调优、排故，方法必

然会有变化，以往的经验有可能不再适合。

还有，以前在 9i 时代，Buffer Cache 池、共享池已经非常完善，其原理基本不会发生变化了。到了 10G，Buffer Busy Waits 就发生了变化。

从 12c 版本开始，在“云应用”的背景下，数据要有更好的流动性，接上插头或打开水管，数据就流出来了，“可插拔数据库”是 Oracle 在这方面的尝试。另外，不得不提是 OGG，虽然它不是在 12C 中新推出的，但它无疑可以增加数据的流动性。

除了让数据“流”起来之外，应用程序升级到“云应用”后，对数据的规模要求一定越来越大，因此大数据是不可避免的趋势。其实使用 Oracle RAC 和 ASM，也可以方便地搭建大数据计算平台。大数据通常都要数据分布化存储，这样才能有更好的扩展性，才能更好地支持大规模数据。

ASM 也能实现数据的分布化存储，扩展也可以很方便。RAC 则可以实现分布式计算，也就是多台主机一起进行计算。比如，先看一张 RAC 的网络架构，如图 9-1 所示。

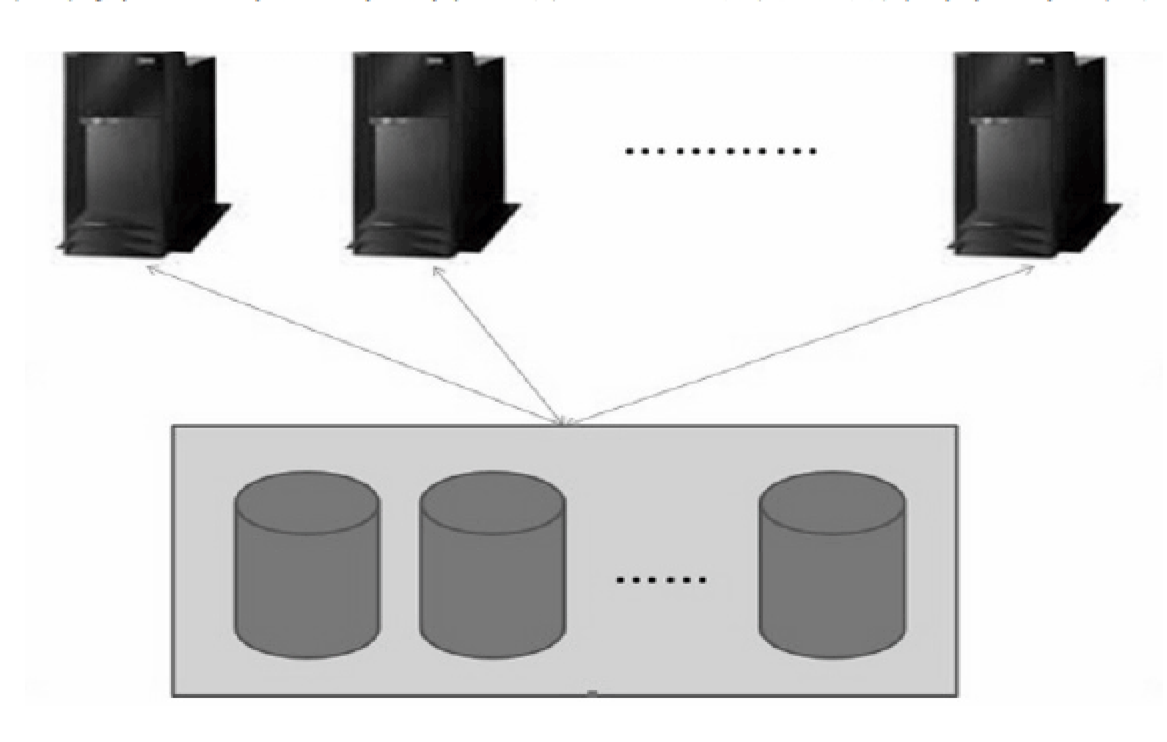


图 9-1 Oracle RAC 架构示意图

图比较简单，上面是 N 个 RAC 节点，下面是一台共享存储，其中有 N 块硬盘。这是 RAC 的传统架构。通过这幅图表达一个问题，先不讨论 RAC 节点数多对性能的影响，单说 RAC 的共享存储，就一直是 RAC 的一大限制。共享存储限制了数据的分布性。像 GreenPlum 和 Hadoop 等都是 Nothing Shared 的全分布式架构，扩展更方便，能力更强。其实使用 iSCSI+ASM，做一个类似分布式的架构也并非难事，如图 9-2 所示。

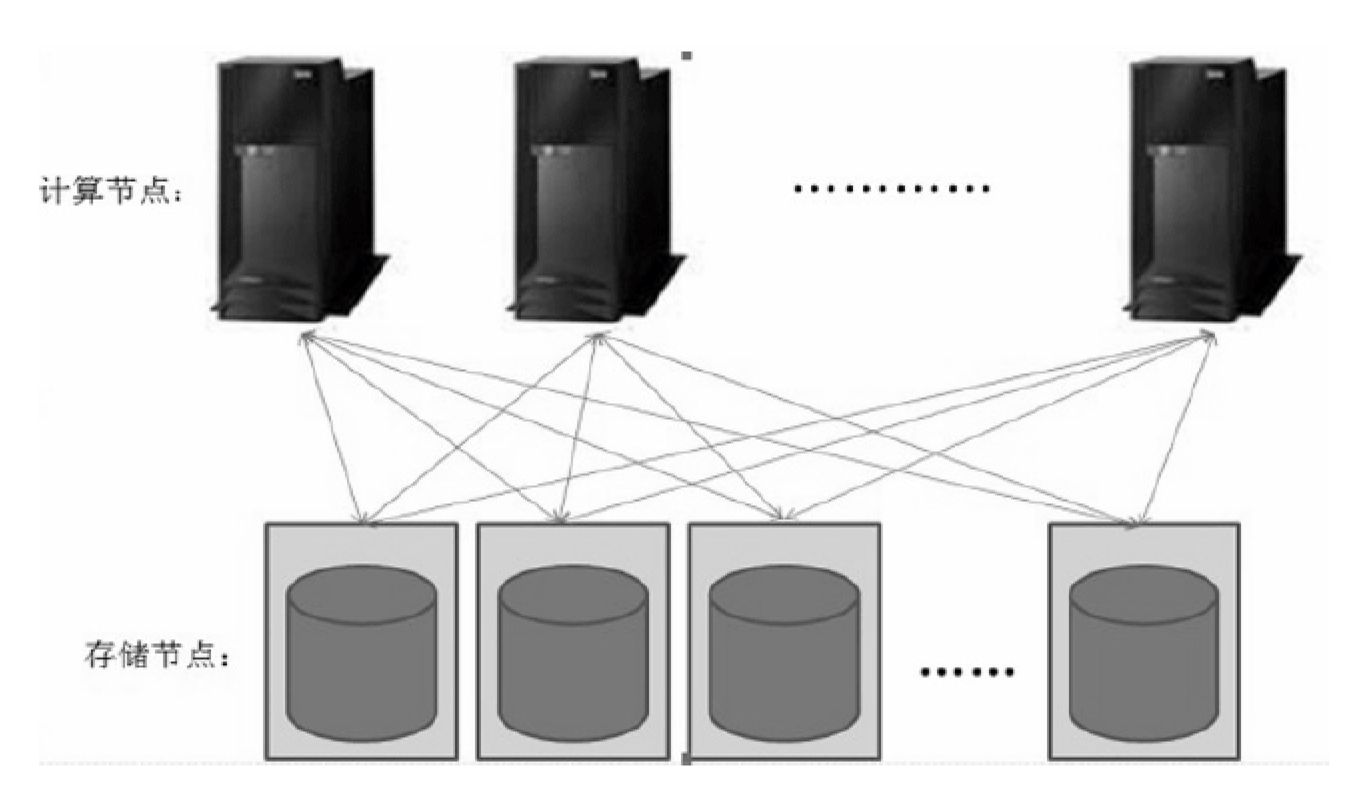


图 9-2 传统数据库集群架构

下面的存储节点可以是有很多硬盘的 PC Server，每个 PC Server 的盘用 iSCSI 技术输送到所有计算节点，然后用这些盘创建 ASM 的 DiskGroup。以后随着数据空间规模的扩大，只要再接入更多的 PC Server，就可以扩展空间，扩容也很方便。不过很多情况下，RAC 的部署模式如图 9-3 所示。

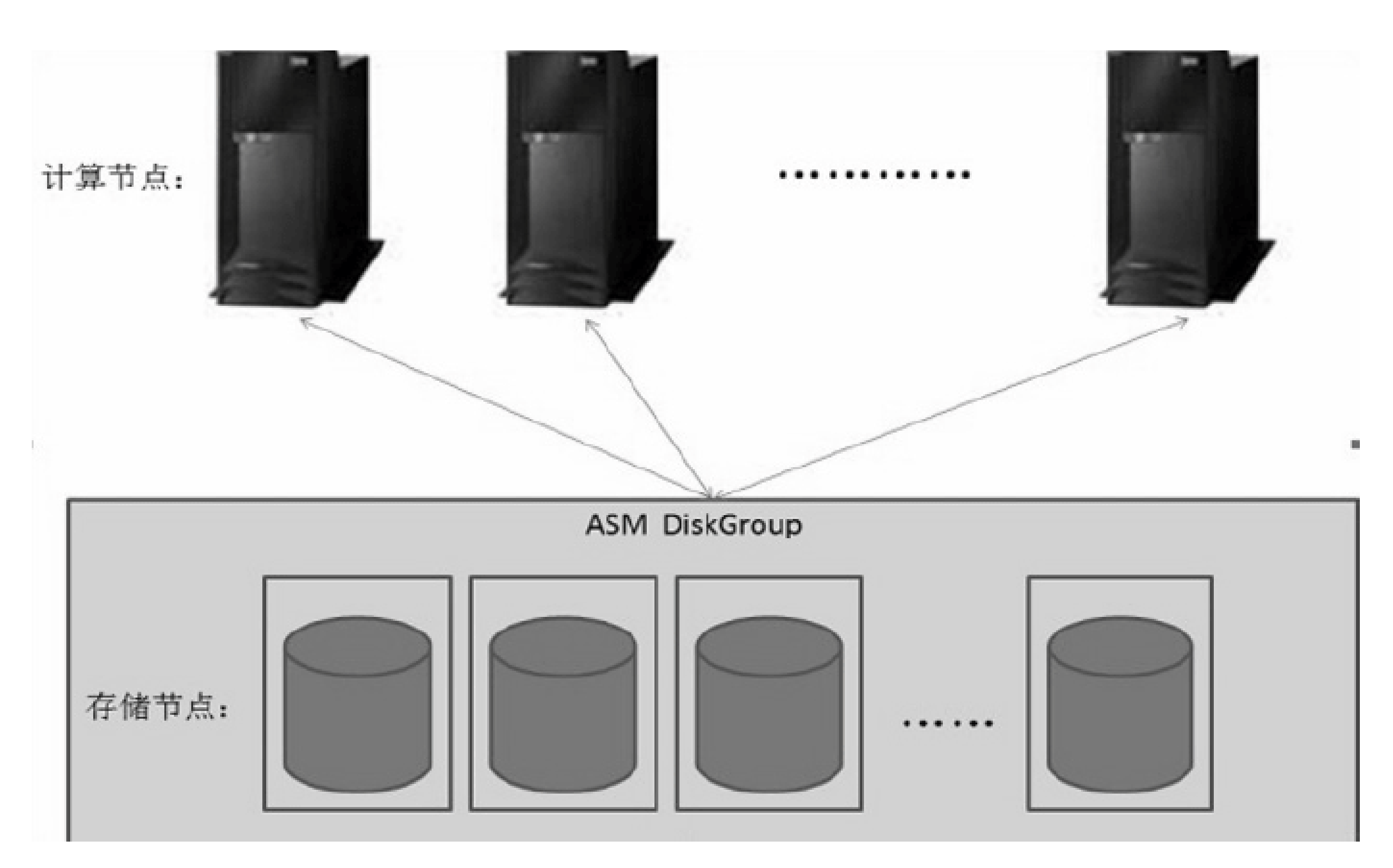


图 9-3　Oracle 数据库集群架构

在图 9-3 中，还是集中式的架构，即一套 ASM，集中存放所有数据。其实数据是分布式的。假设在 ASM 中建一个表，在 AU 大小为 1MB 的情况，这个表的数据会以 1MB 为单位，分布到所有磁盘中，如图 9-4 所示。

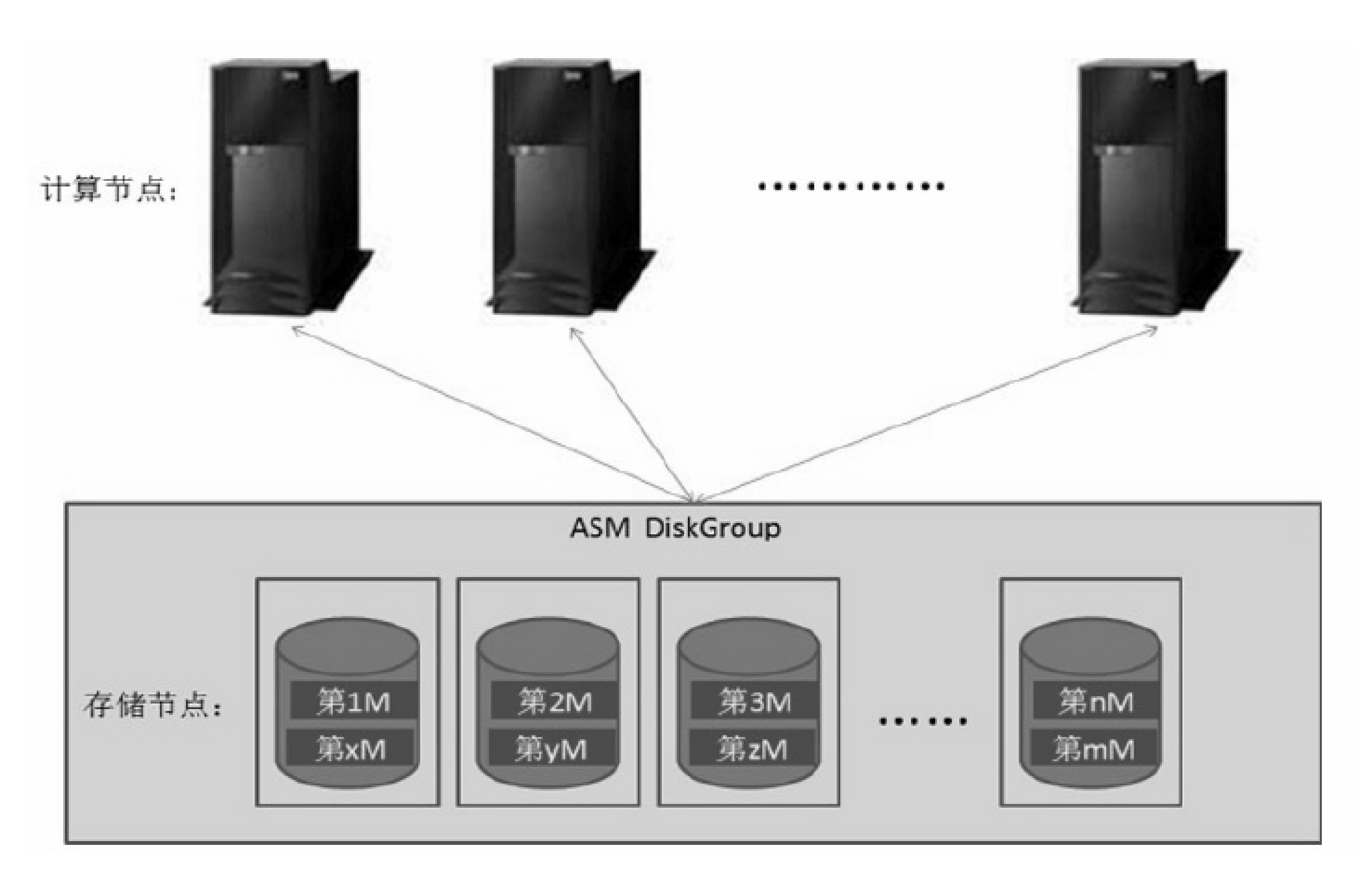


图 9-4　ASM 管理集群共享存储

假设在节点 1 发起一条命令，对这个表进行全扫描并排序，在 RAC 下并行操作是会被传播到其他节点的。也就是说，图 9-4 中的每个计算节点都会一起完成扫描、排序。
在协调节点的控制下，全扫描、排序将按图 9-5 的方式完成。

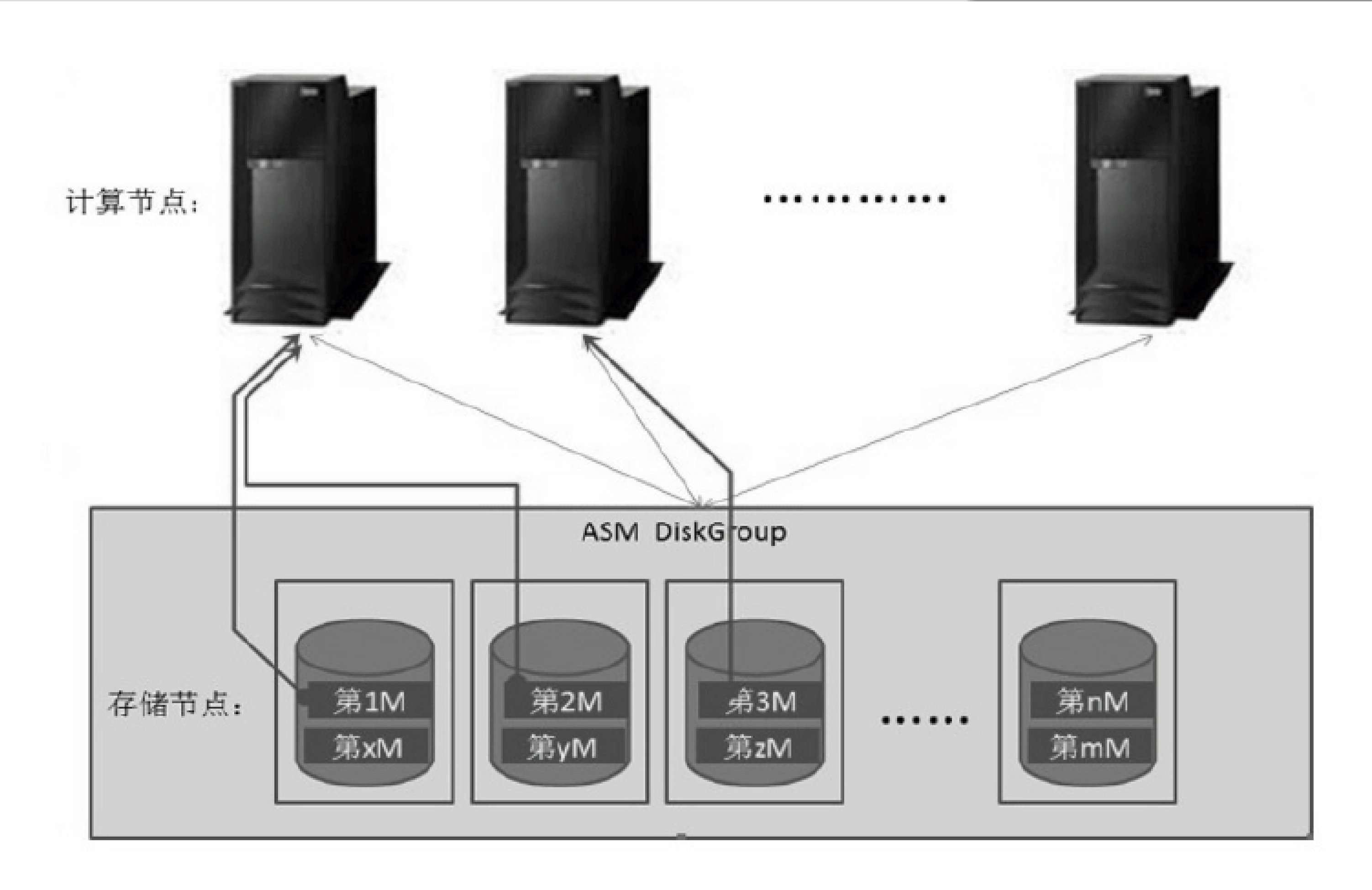


图 9-5　ASM 管理集群存储

节点 1 从存储节点 1、2 中读取第 1、2MB。同时节点 2 从存储点 3 中读第 3MB，等等。每个计算节点会同时从多个存储节点一起读数据。多个计算节点，每个都在对一部分数据做操作。每个存储节点都在为不同计算节点输送数据，如图 9-6 所示。

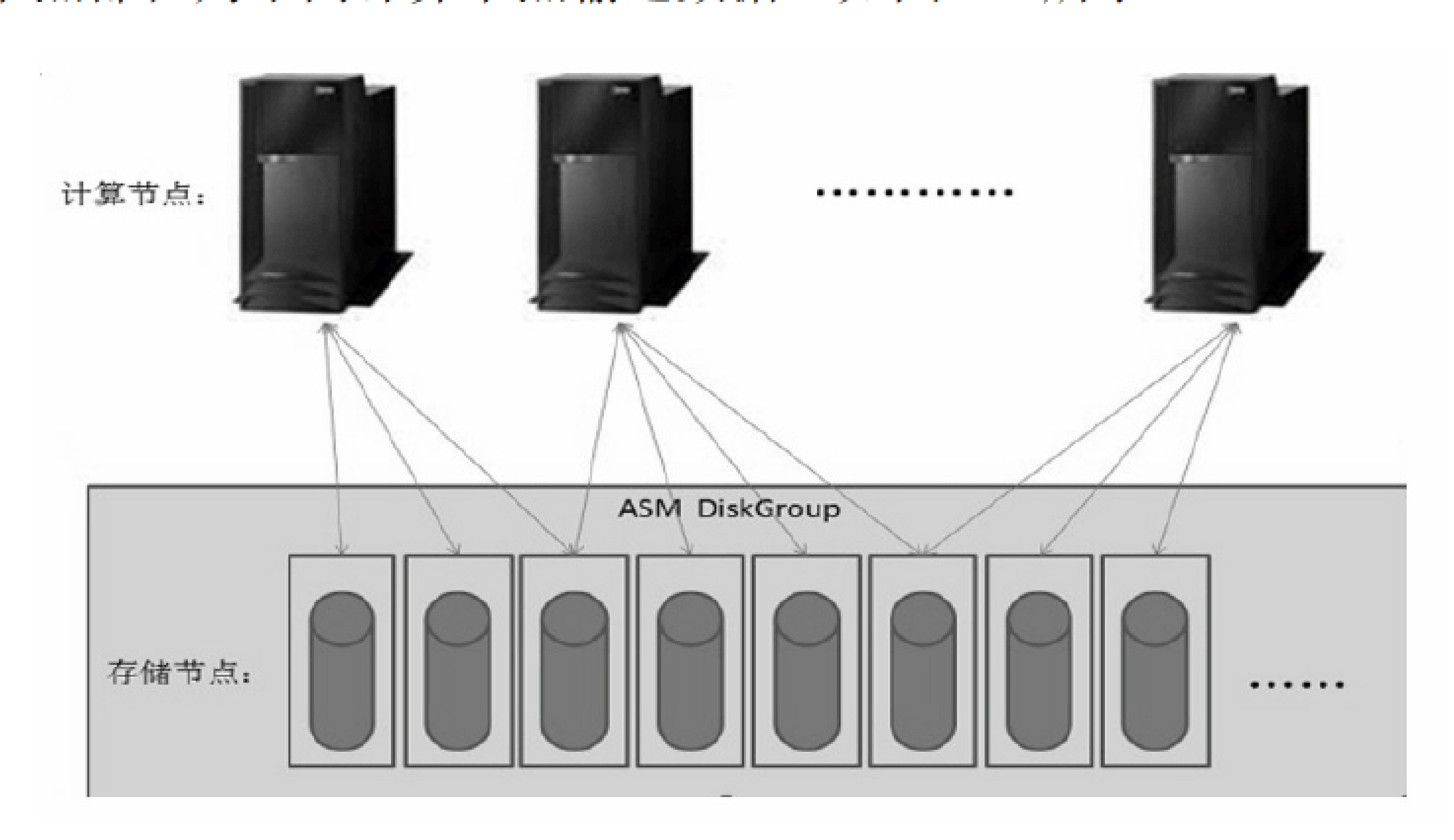


图 9-6　ASM 存储数据分布存储示意图

数据分散存储在多台 PC Server 中，这就是一套分布式的系统，但同时它又是一套集中式的数据库系统。试想一下，如果下面的存储节点使用了 SSD 技术，网络使用和光纤有一比的 InfiniBand，这套系统可以提供超大的 IOPS 和超高 IO 吞吐量。并不一定只有 GreenPlum、Hadoop 等才是大数据，ASM、RAC 也可以实现分布化、大数据。而且，由于对外还是一个统一、集中的数据库，因此这样的方案还有纯分布式不能提供的好处。还有一个问题，即 RAC 节点数太多的影响。其实 RAC 节点多少对整体性能影响不大。节点太多，主要影响稳定性。但 Oracle 一直在改善多节点的稳定性，相信未来多节点不会是 RAC 稳定性的瓶颈。

以上内容只是从一个方面阐述了对于 Oracle 来说可以采用何种方式来实现分布式架构的，但是分布式的架构还不是最终 Oracle 希望呈现给用户的功能，下面将介绍 Oracle 12c 版本的多租户功能。

9.2 什么是多租户功能

本节将重点介绍 Oracle 12c 版本的一项重大革新功能，即多租户新特性，这里涉及很多关于这个多容器数据库的材料。从本节开始读者将会经常看到 CDB 这个英文缩写，全称为 Container Database，即 Oracle 容器数据库；还有另外一个名词 PDB，即 Oracle 可插拔数据库。通过本节的学习，读者将会了解什么是可插拔的数据库、何时需要使用多租户容器数据库、目前这个版本的 Oracle 12c 多租户特性与非容器数据库有哪些不同之处，以及如何创建和维护这些容器数据库（或者说如何有效地管理它）。

9.2.1 Oracle 12c 多租户功能简介

Oracle 12c 的可插拔数据库功能目前也是 Oracle 公司的云控件，通过数据库层面的虚拟化，实现了统一管理、资源整合和快速部署等功能，并且 Oracle 12c 版本的资源管理器组件取代了旧的独立 OEM，可能会实现云数据库的管理。

借助可插拔数据库技术，Oracle 12c 在数据库层而非应用层支持多租户，可以在单一实体机器中部署多个数据库。注意，这里说的部署多套数据库并非是部署多个数据库实例，而是在同一个数据库实例中，以服务的形式部署多个数据库，也叫 PDB，不同的 PDB 共享同一个实例的所有计算资源，并且互相之间具有隔离机制，对服务器 vmware 虚拟化了解的读者可以类比理解，形式上和服务器的虚拟化有些类似。每个数据库都能以动态插拔的方式，在多租户架构下扩充、整合、升级与备份，从而避免了传统应用部署所需的冗长步骤，并降低运行成本和硬件资源的需求。

9.2.2 容器数据库介绍

Oracle 的多租户和 MySQL、MSSQL 类似，把之前的一个实例对一个数据库的情形（RAC 是多个实例对一个数据库）整合成了一个实例下可以挂多个数据库，并且定义为可插拔的，听起来很炫。就像在没有多租户特性之前，Oracle 与 MSSQL 以及 MYSQL 还是有很大的差异，因此对于 Oracle 的多租户也有一些不同的地方。

Oracle 多租户环境包含一个容器数据库（CDB）和零个或多个可插拔数据库（PDB），一个 PDB 是一个模式、模式对象以及非模式对象，如到一个 Oracle 网络客户端作为非 CDB。Oracle 12c 之前的版本都是非 CDB 数据库，如图 9-7 所示。

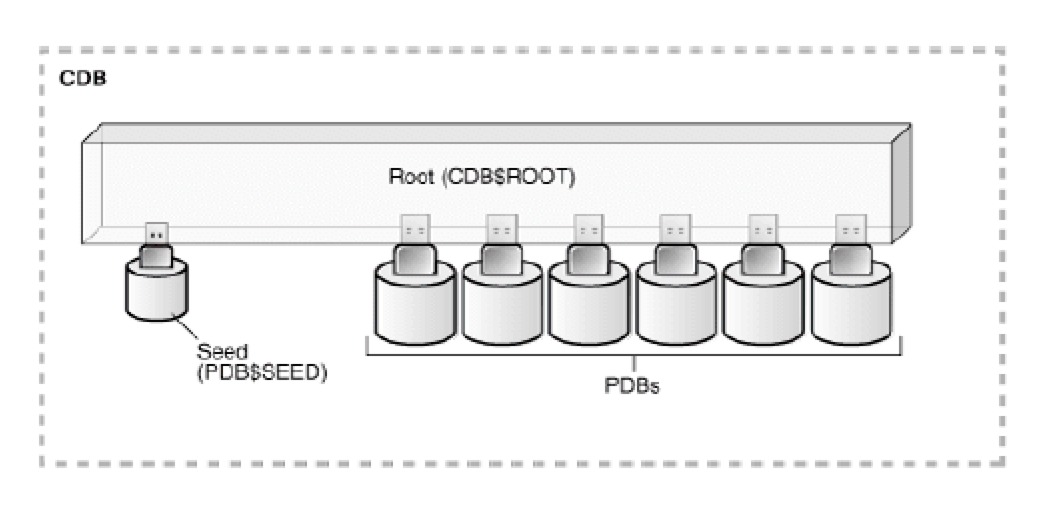


图 9-7　容器数据库

每个 CDB 都有以下容器：CDB 根容器（也简称根）和 PDB 容器（也简称系统容器）。CDB 根是一个模式，是所有 PDB 的非模式对象的集合。根容器存储 Oracle 提供的元数据和公共用户。元数据的一个例子是 Oracle 提供的 PL/SQL 包的源代码。公共用户是每个容器中已知的数据库用户。根容器被命名为 CDB$ROOT。系统容器包括根 CDB 和在 CDB 所有的 PDBS。因此，系统容器是 CDB 本身的逻辑容器。一个 PDB 包含特定功能集所需的数据和代码。例如，PDB 可以支持特定应用，如人力资源或销售应用。可以根据业务需求添加 PDB。

PDB 属于零个或一个应用程序容器。如果 PDB 属于应用程序容器，那么它是一个应用程序 PDB。例如，cust1_pdb 与 cust2_pdb 应用 PDBS 可能属于该 saas_sales_ac 应用程序容器，在这种情况下，它们不属于任何其他应用程序容器。应用程序种子是可选的应用程序 PDB，用作用户创建的 PDB 模板，便于快速创建新的应用程序 PDB。

种子 PDB 是 CDB 可用于创建新的 PDB 的系统提供的模板。种子 PDB 命名为 PDB$SEED。不能添加或修改对象 PDB$SEED。

9.2.3　多租户功能的优势

在 Oracle 12c 版本中，多租户的功能特性相对于以前的数据库版本或者与其他同类型关系型数据库相比有如下优势。

- 整合密度高，在 12c 多租户架构的新特性中，一个多租户容器数据库中的多个可插拔数据库共享内存和后台进程。相比于旧架构，这样可以整合更多的可插拔数据库，且提供与基于 schema 的整合类似的优点，但避免了该方法所需的重大应用程序更改。
- 快速克隆，使用新的 SQL 命令，创建可插拔数据库、在容器间移动可插拔数据库和克隆可插拔数据库只需几秒钟的时间。当底层文件系统支持瘦供应时，只需在 SQL 命令中使用关键字“snapshot copy”，几乎瞬间即可克隆 TB 级数据。
- 快速打补丁和升级，只需投入修补一个多租户容器数据库的时间和精力，即可修补所有多个可插拔数据库。要修补一个可插拔数据库，只需将其拔/插到不同的 Oracle Database 软件版本中的多租户容器数据库，就避免了老版本数据库中当需要修复补丁时需要进行多次重复工作，有多少数据库基本就需要打多少次补丁。
- 集中管理，通过将现有数据库整合为可插拔数据库，管理员可以将多个数据库作为一个管理。例如，在多租户容器数据库级别执行备份和灾难恢复等任务。DBA 只需要维护一个数据库即可。
- 资源管理，相比于 12c 之前的版本，Oracle Database 12c 中对资源管理器进行了扩展，加入了特定功能来控制多租户容器数据库中可插拔数据库之间的资源竞争。

9.2.4　创建多租户数据库

在使用 DBCA 创建一个多租户数据库之前，需要引入 CDB 和 PDB 的概念，CDB 即 Container Database，容器数据库，是作为多租户架构数据库中的容器存在的。从字面就可以理解，它是作为容纳整合后的其他数据库的一个容器而存在的。PDB 即 Pluggable Database，可

以连接为 12c 中独立的数据库，但是现在是整合到了一个 CDB 容器里面，如图 9-8 所示。

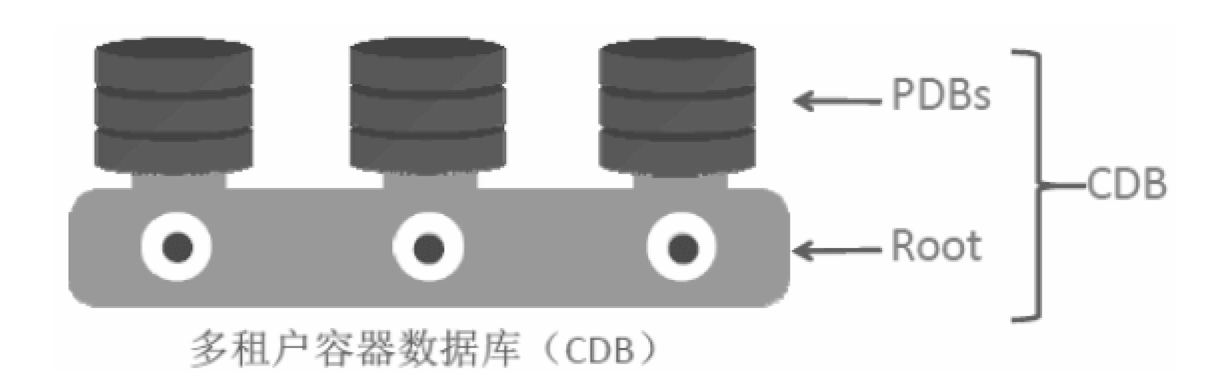


图 9-8　PDB 与 CDB 的关系

在多租户数据库的架构中，有一个多租户容器数据库，即 CDB。CBD 中又有多个 PDB，PDB 可以理解为 12c 之前的独立数据库。

下面使用 DBCA 创建一个多租户数据库，本例中为 2 节点 RAC 环境，版本为 12.1.0.2。

（1）参考图 9-9，在 DBCA 创建数据库的过程中，选中 Create As Container Database，即创建容器数据库，本例中创建一个容器数据库，且里面包括一个 PDB 可拔插数据库，名字为 PDB_TEST1。如果还需要创建多个可拔插数据库，可以在这里直接选择，也可以在事后再进行创建。

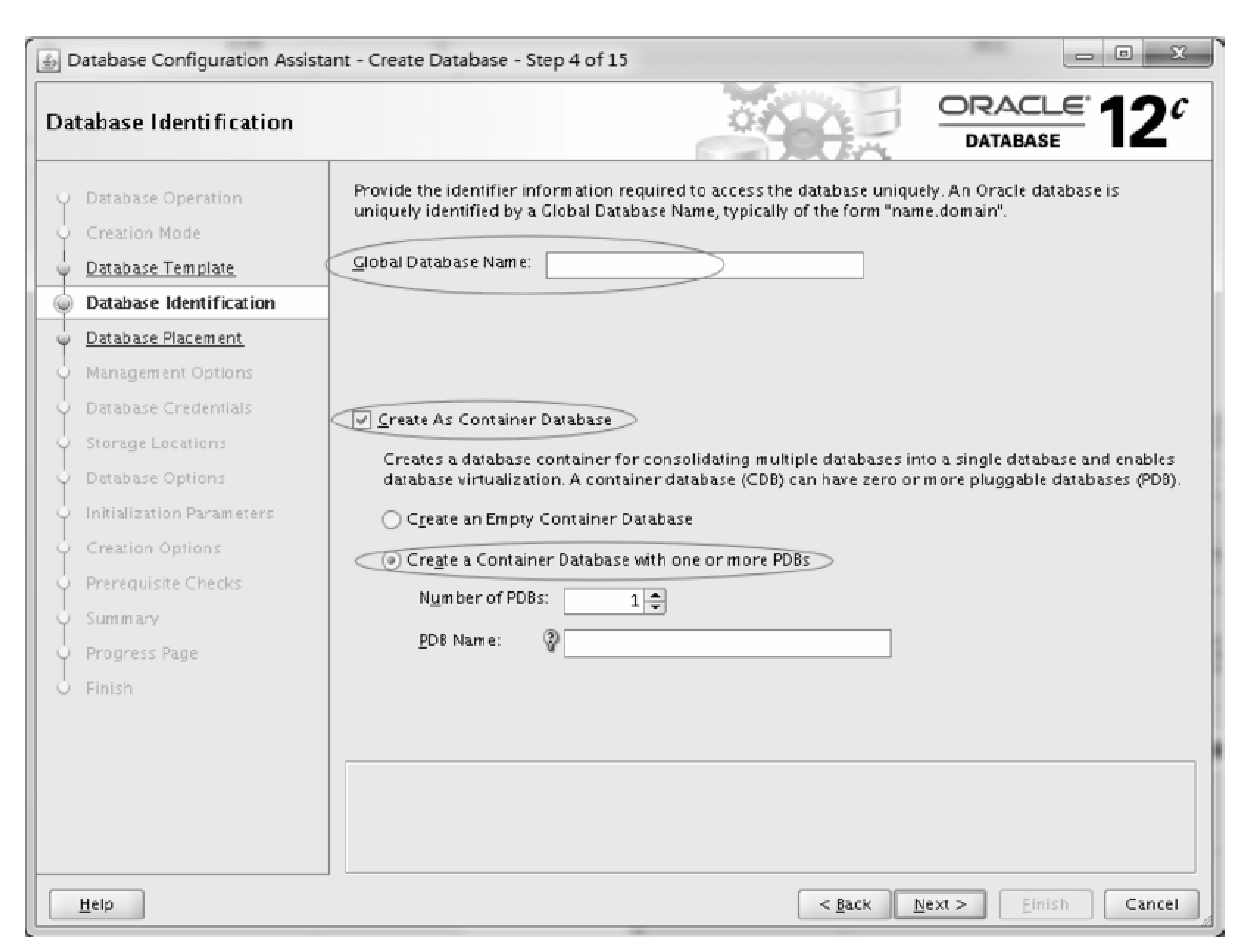


图 9-9　DBCA 创建 PDB

（2）参考图 9-10，创建 Server Pool 资源池，本例仅仅需要 2 节点 RAC，Cardinaliry 选择 2 就好。在有更多节点的情况下，可以创建多个资源池，并且资源池可用节点等都可以按照实际情况来进行具体规划。在做数据库 PAAS 云时，可以详细规划，本例中不做更详细的描述。

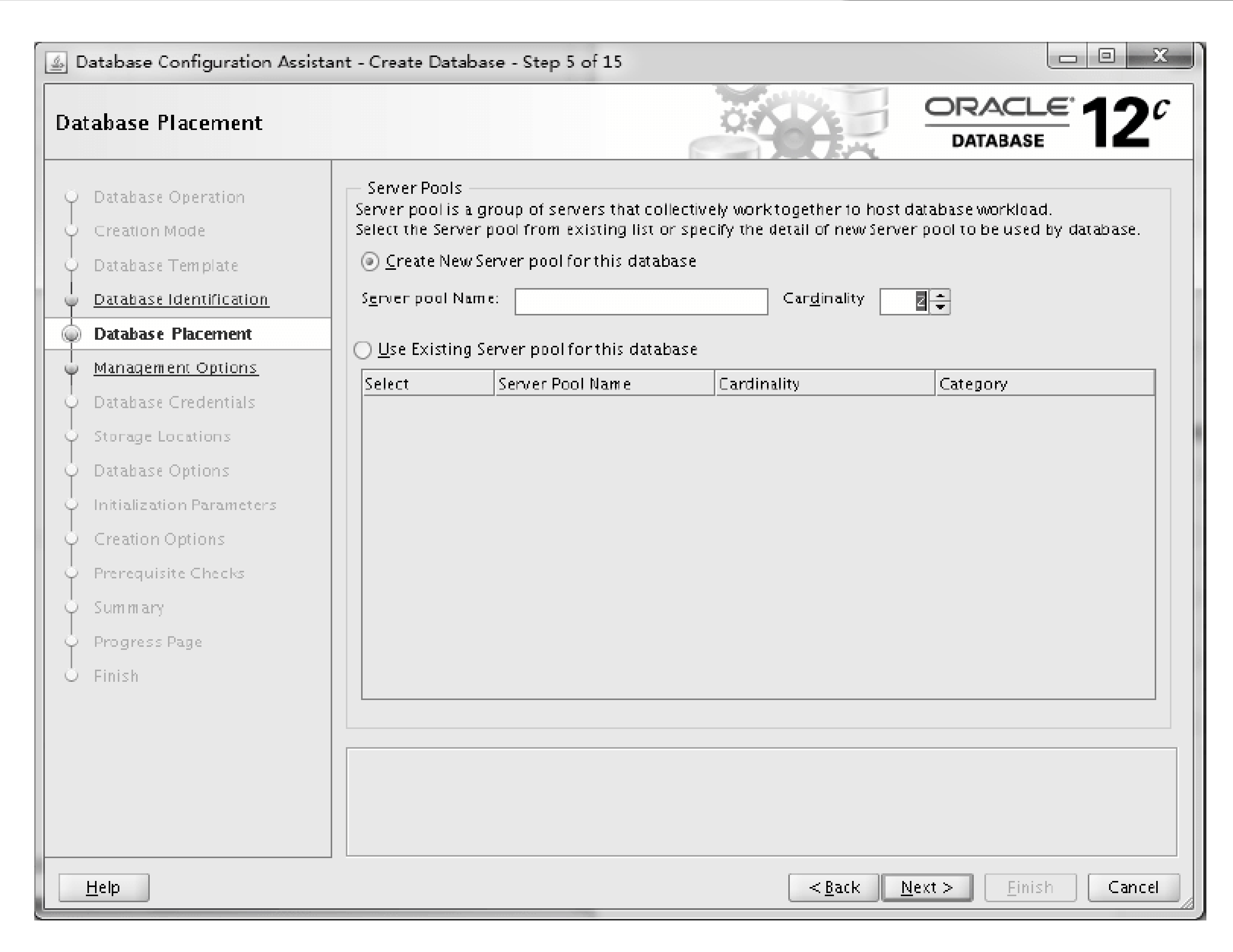


图 9-10　DBCA 指定资源池配置

（3）DBCA 其他步骤不再讲解，和普通数据库创建过程是一致的，在数据库创建完毕后，查看集群服务状态：

```
[grid@node2 ~]$ crsctl stat res -t
------------------------------------------------------------------------------
--
Name..... Target.State....Server......... State details...
------------------------------------------------------------------------------
--
Local Resources
------------------------------------------------------------------------------
--
ora.CRS.dg
....... ONLINE.ONLINE... node1.........STABLE
....... ONLINE.ONLINE... node2.........STABLE
ora.DATA.dg
....... ONLINE.ONLINE... node1.........STABLE
....... ONLINE.ONLINE... node2.........STABLE
ora.FRA.dg
....... ONLINE.ONLINE... node1.........STABLE
....... ONLINE.ONLINE... node2.........STABLE
ora.LISTENER.lsnr
....... ONLINE.ONLINE... node1.........STABLE
....... ONLINE.ONLINE... node2.........STABLE
ora.asm
....... ONLINE.ONLINE... node1.........Started,STABLE
....... ONLINE.ONLINE... node2.........Started,STABLE
ora.net1.network
....... ONLINE.ONLINE... node1.........STABLE
....... ONLINE.ONLINE... node2.........STABLE
ora.ons
....... ONLINE.ONLINE... node1.........STABLE
```

```
....... ONLINE.ONLINE... node2.........STABLE
-------------------------------------------------------------------------------
--
Cluster Resources
-------------------------------------------------------------------------------
--
ora.LISTENER SCAN1.lsnr
...1....ONLINE.ONLINE... node1.........STABLE
ora.MGMTLSNR
...1....ONLINE.ONLINE... node1.........169.254.230.82 20.20
............................... .20.1, STABLE
ora.cvu
...1....ONLINE.ONLINE... node1.........STABLE
ora.mgmtdb
...1....ONLINE.ONLINE... node1.........Open, STABLE
ora.node1.vip
...1....ONLINE.ONLINE... node1.........STABLE
ora.node2.vip
...1....ONLINE.ONLINE... node2.........STABLE
ora.oc4j
...1....ONLINE.ONLINE... node1.........STABLE
ora.scan1.vip
...1....ONLINE.ONLINE... node1.........STABLE
ora.test.db
...1....ONLINE.ONLINE... node2.........Open,STABLE
...2....ONLINE.ONLINE... node1.........Open,STABLE
```

（4）创建了一个 CDB 容器数据库，且里面包含一个 PDB，在该 CDB 中再创建一个 PDB。这里使用 DBCA 工具来创建。如图 9-11 所示，选择 Manage Pluggable Databases。

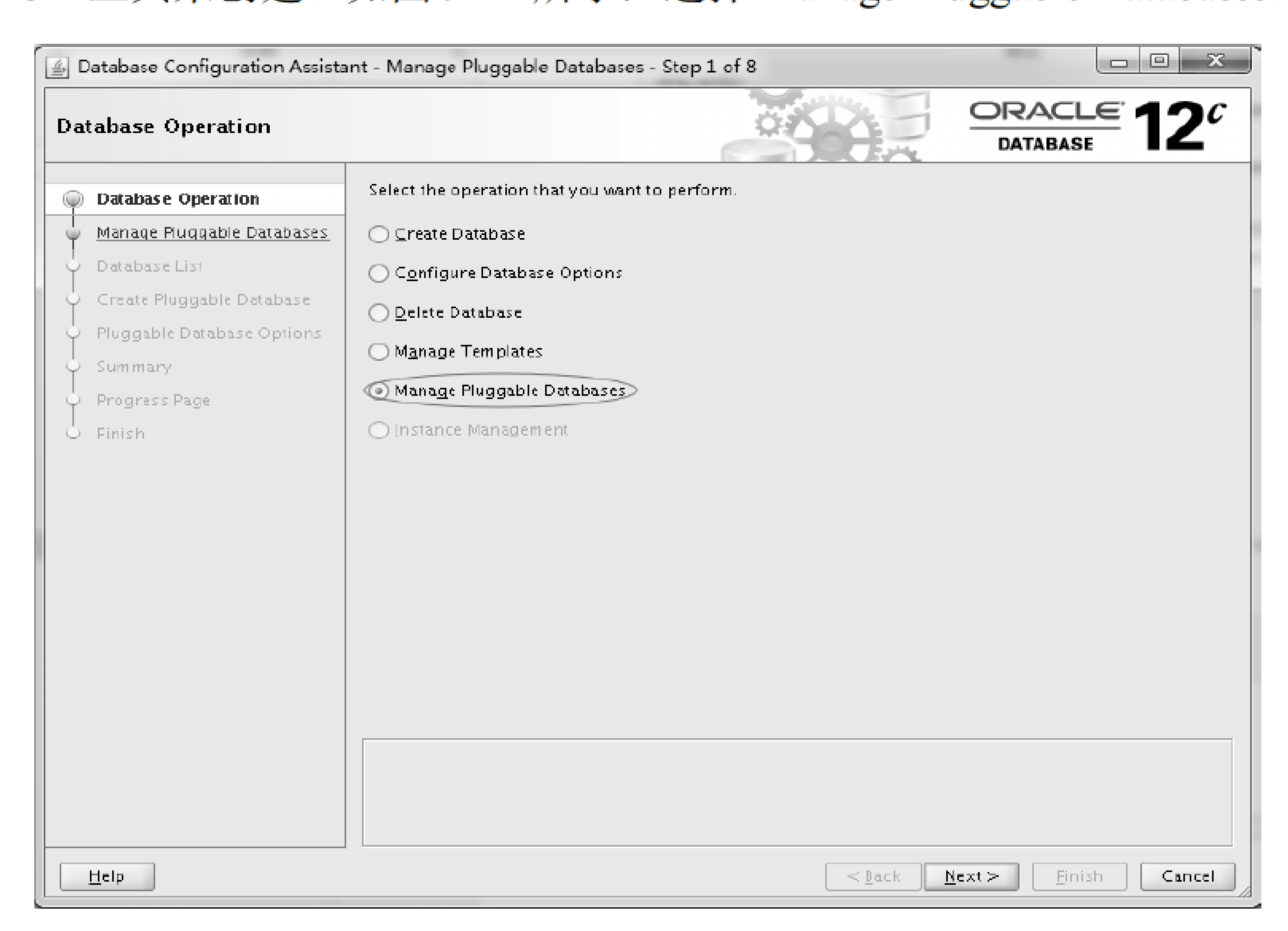


图 9-11　DBCA 管理 PDB

（5）单击 Next 按钮，在下一步中选择 Create a Pluggable Database 来创建一个 PDB，如图 9-12 所示。

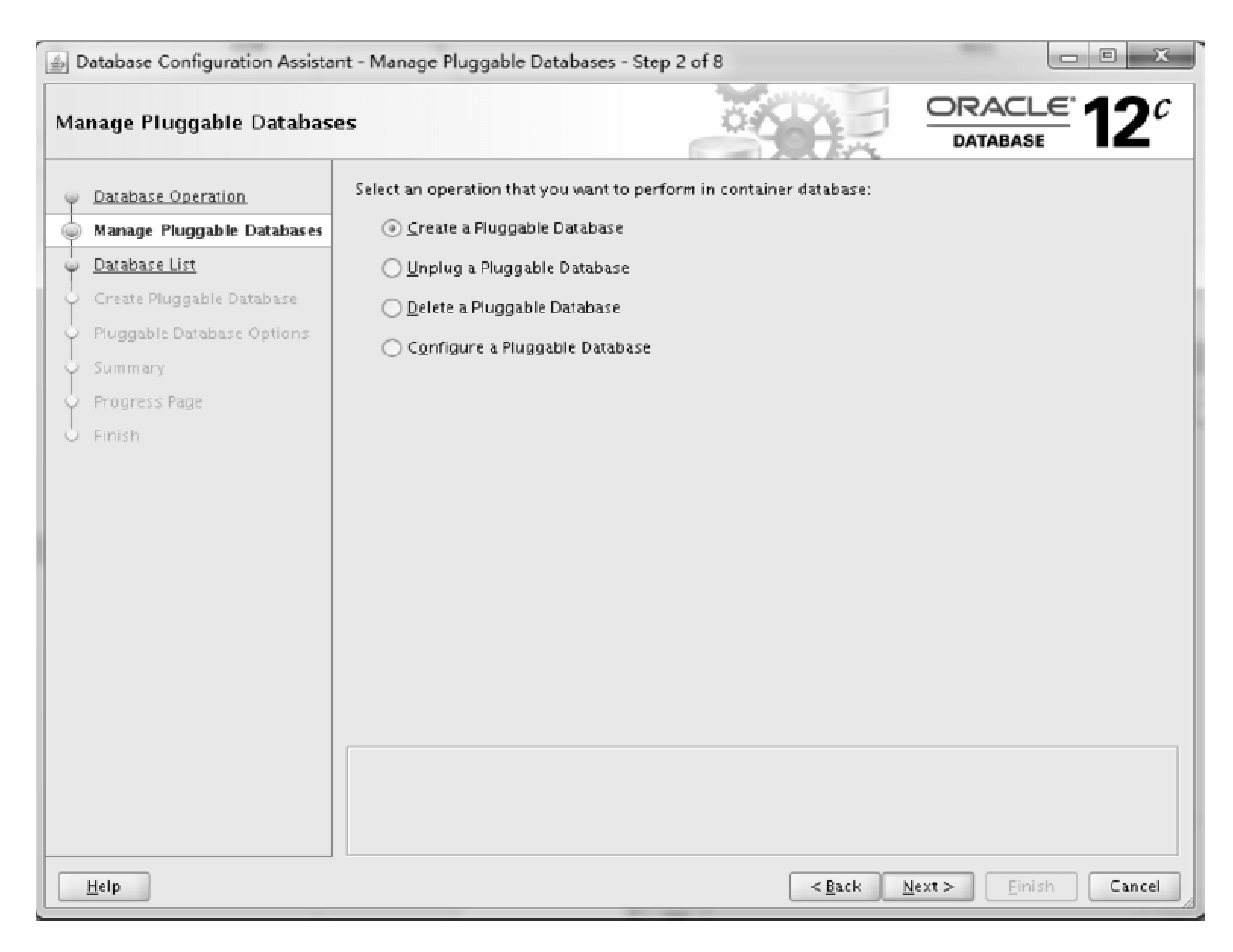


图 9-12　创建 PDB

（6）选择在哪个 CDB 上创建 PDB。由于之前只创建了一个 CDB，因此这一步默认即可，见图 9-13。确认信息无误之后，单击 Next 按钮。

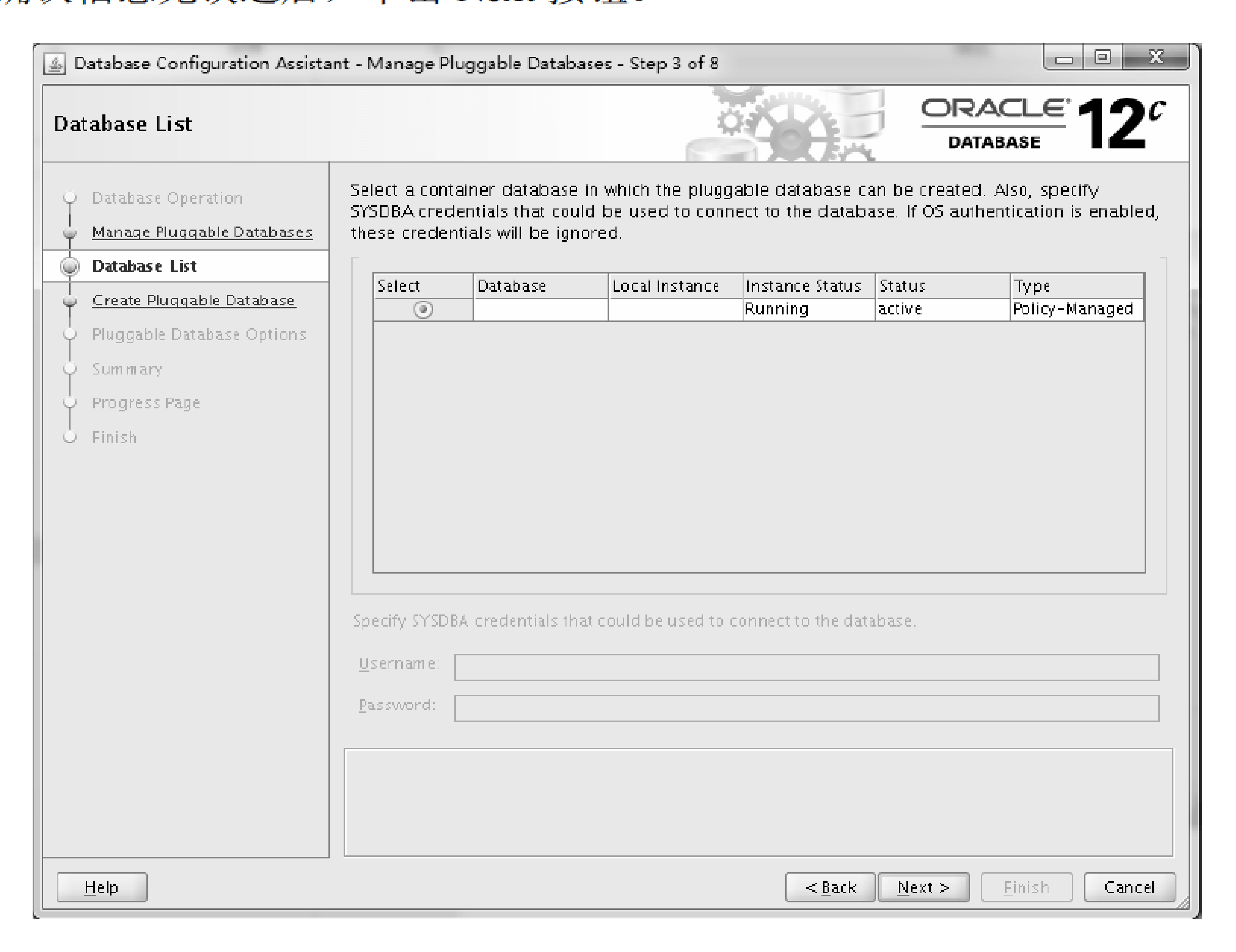


图 9-13　PDB 创建选项

（7）选择 Create a new Pluggable Database，创建一个新的可拔插数据库，见图 9-14。

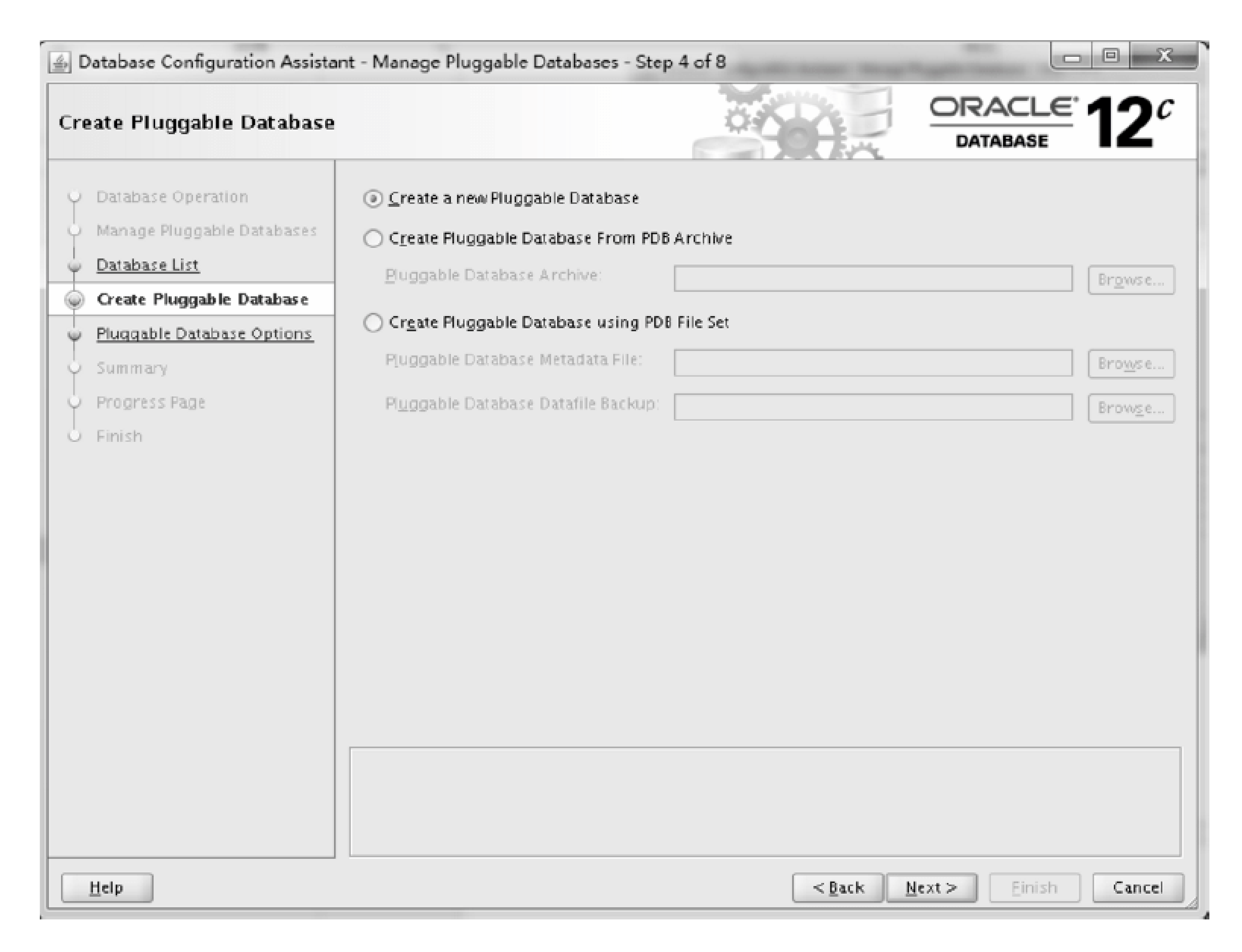


图 9-14　创建可插拔数据库

（8）如图 9-15 所示，新创建的 PDB 名称请读者根据自己所需填写，这里没有任何的限制和约束。在该界面中还可以创建默认的表空间与数据存储，数据存放到本例中是存放在+DATA 磁盘组中，创建该 PDB 的管理员名称，同样也是根据用户需求填写即可。所有信息填写完毕之后，单击 Next 按钮进入下一步。

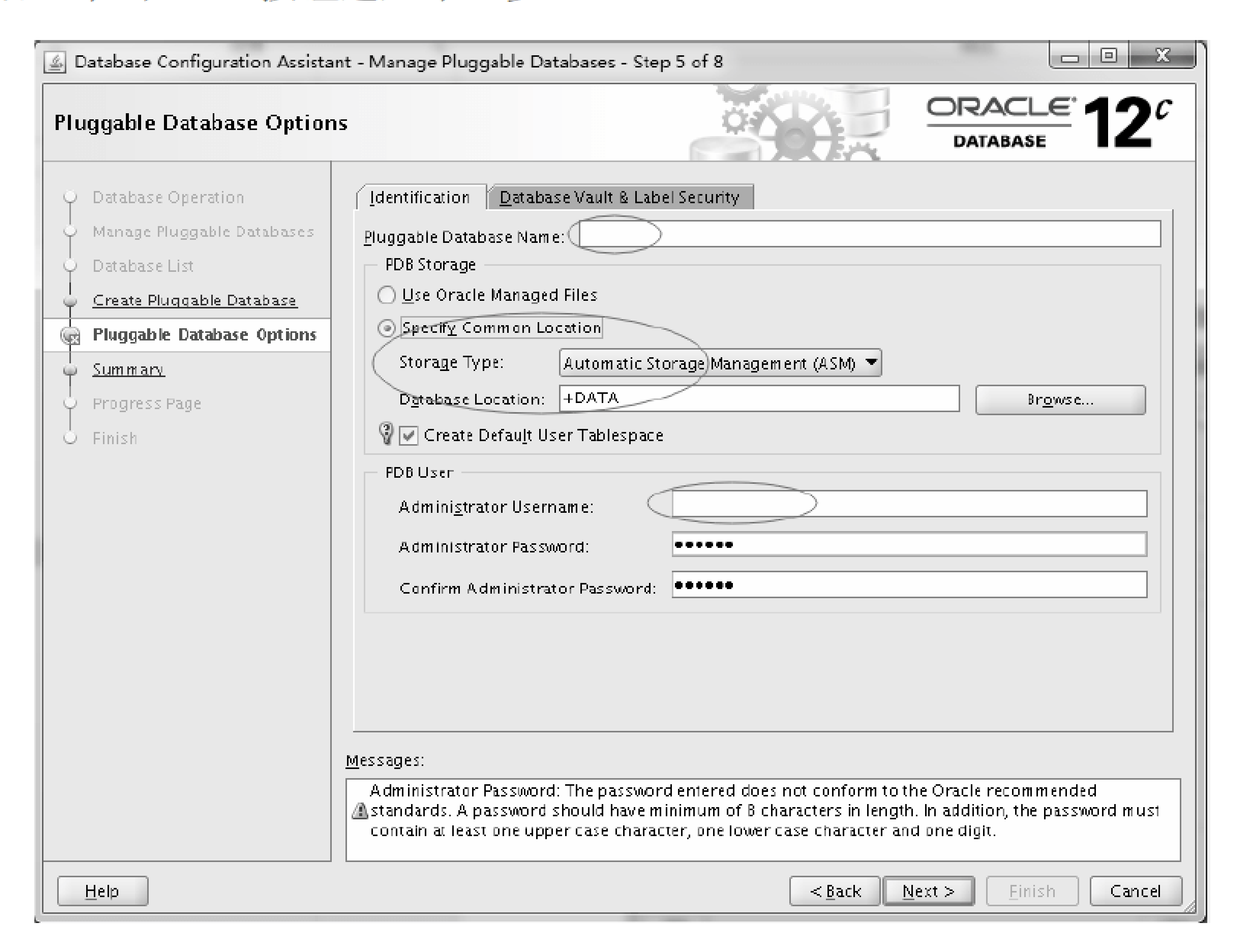


图 9-15　为 PDB 命名

（9）该界面是用户确认步骤，这里会列出我们之前填写的 PDB 信息。用户确认所有信息无误之后，单击 Finish 按钮（见图 9-16），完成创建操作。

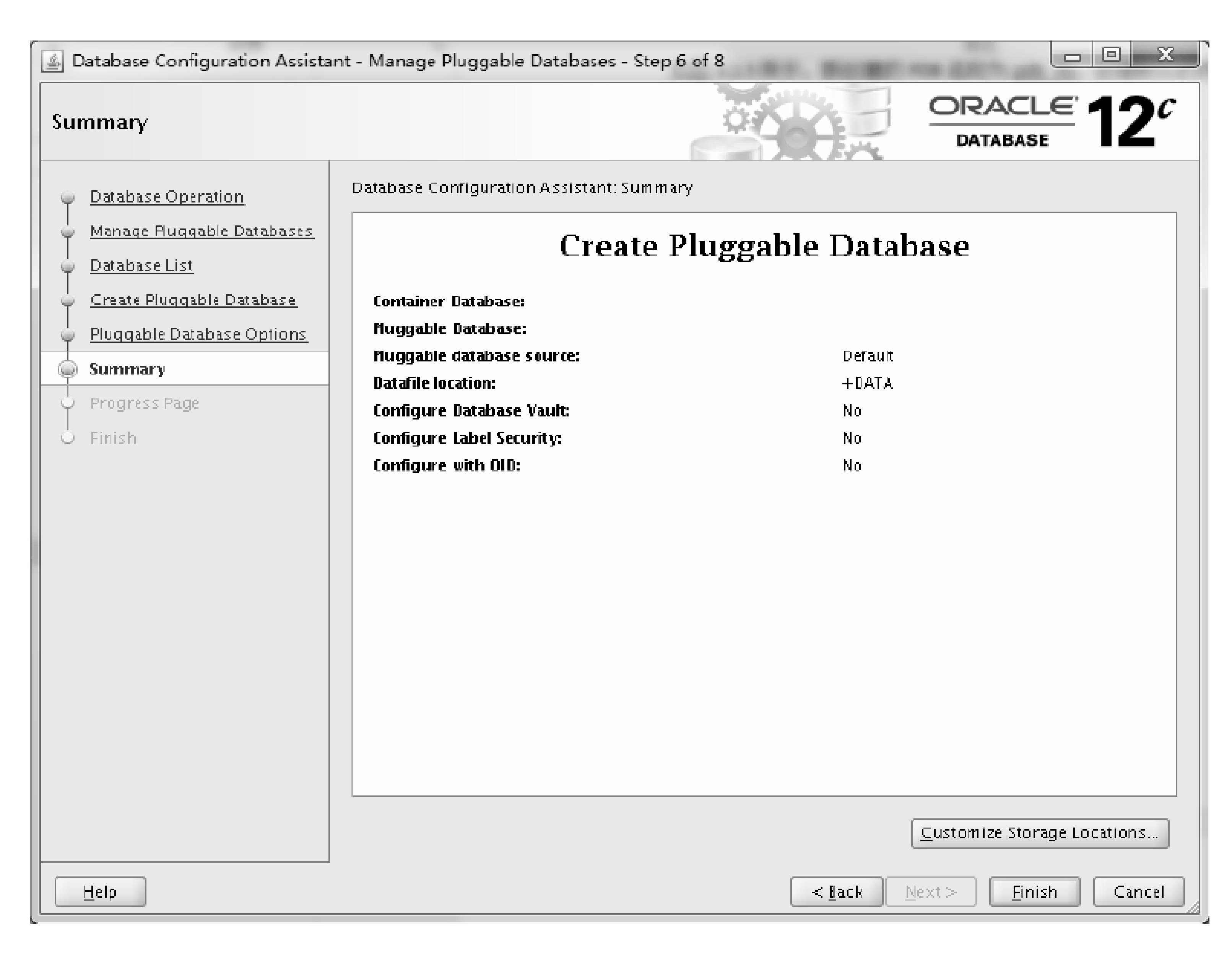


图 9-16　完成创建配置

（10）在用户完成创建配置之后，进入真正创建 PDB 的过程，如图 9-17 所示。安装程序会执行 PDB 创建，并显示创建的具体进度，这里请用户耐心等待完成。

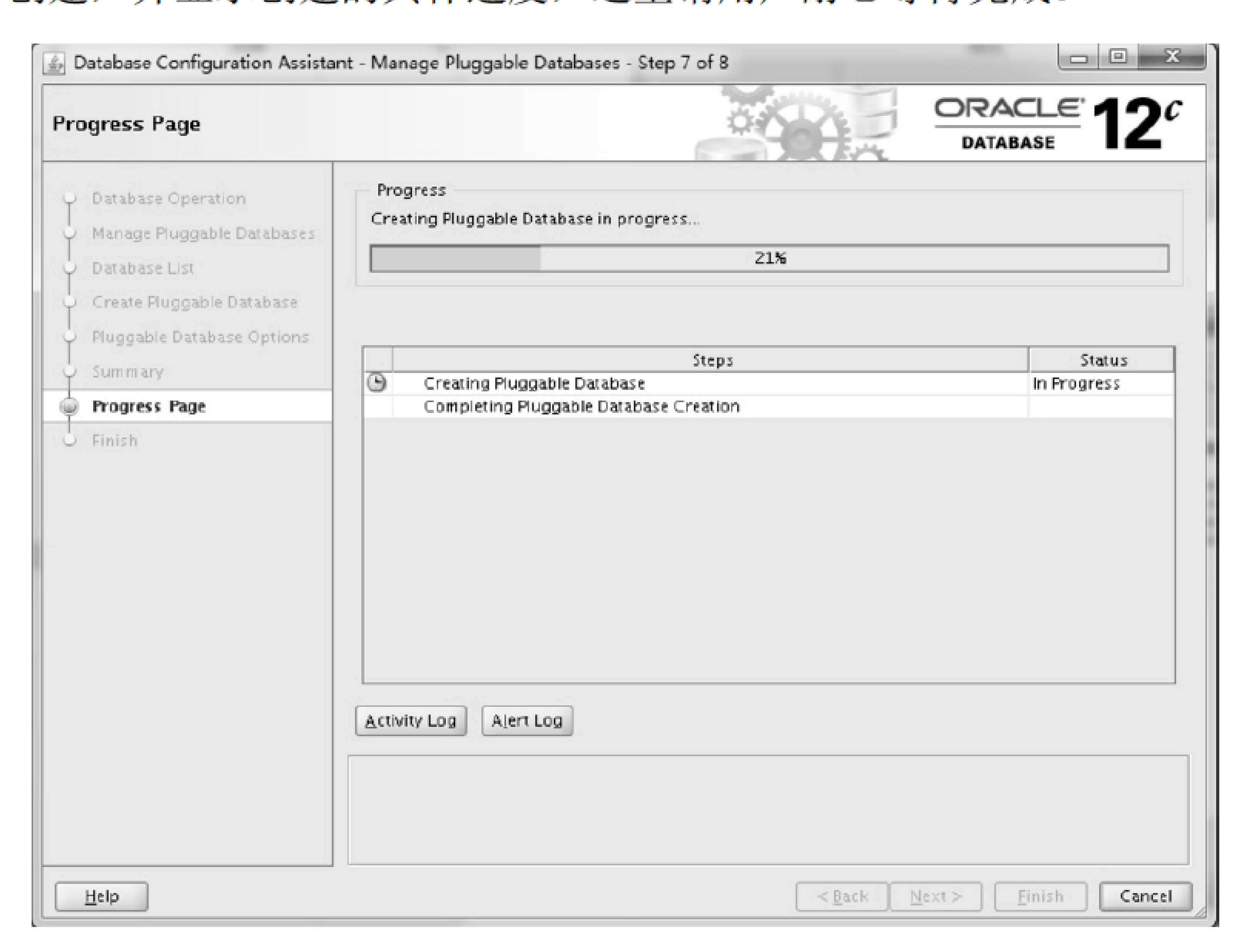


图 9-17　开始创建 PDB

（11）经过几分钟的创建过程，如图 9-18 所示，用户所需的 PDB 创建完毕。这时在 CDB 中已经存在可供使用的 PDB 服务，全部的创建过程都已经执行完毕，请单击 Close 按钮关闭 DBCA 工具。

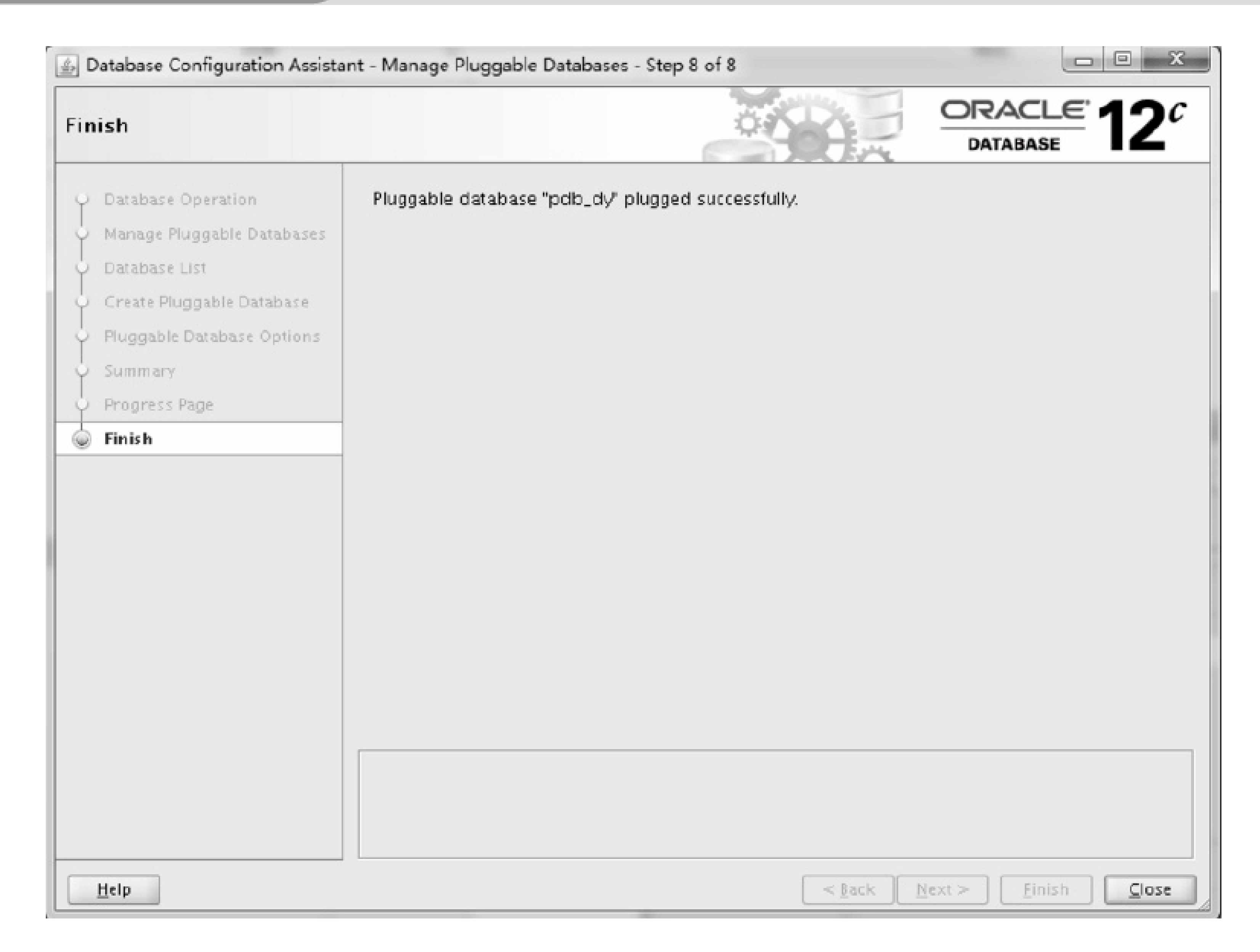
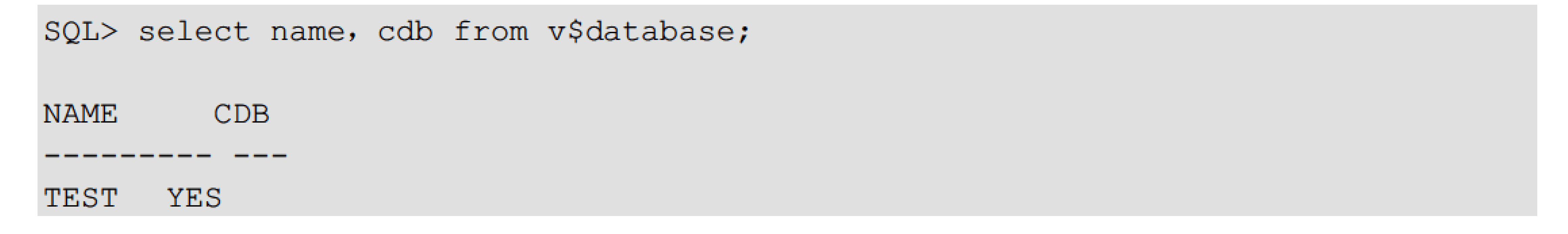


图 9-18　完成创建

9.2.5　基本管理

日常的数据库管理涉及的工作量不是很大，主要集中在多租户环境下的数据库状态查看、启停、打开、关闭等操作上，方法如下：

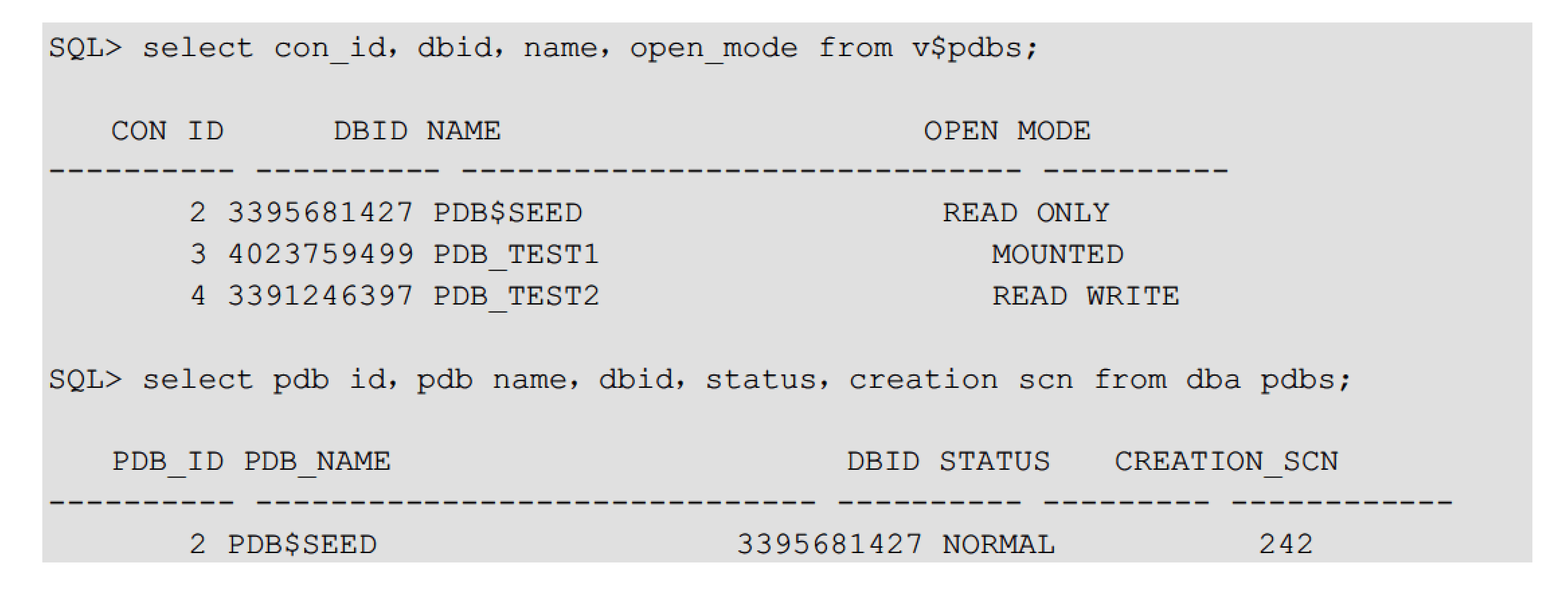
查看当前数据库是否为多租户数据库，v$database 的 CDB 字段显示 YES，表明当前数据库为一个 CDB 数据库：



```
SQL> select name，cdb from v$database;

NAME      CDB
--------- ---
TEST   YES
```

查看当前 CDB 数据库中 PDB 的状态：



```
SQL> select con_id，dbid，name，open_mode from v$pdbs;

   CON ID       DBID NAME                           OPEN MODE
---------- ---------- ------------------------------ ----------
         2 3395681427 PDB$SEED                        READ ONLY
         3 4023759499 PDB_TEST1                          MOUNTED
         4 3391246397 PDB_TEST2                          READ WRITE

SQL> select pdb id，pdb name，dbid，status，creation scn from dba pdbs;

   PDB_ID PDB_NAME                             DBID STATUS    CREATION_SCN
---------- ------------------------------ ---------- --------- ------------
         2 PDB$SEED                       3395681427 NORMAL             242
```

```
        3 PDB_TEST1                      4023759499 NORMAL       1449553
        4 PDB_TEST2                      3391246397 NORMAL       1676453
```

可以看到数据库中当前存在两个 PDB，分别为 PDB_TEST1 和 PDB_TEST2，它们的状态一个为 mounted，一个为 redad wirte。另外还有一个 PDB$SEED，在这里可以理解为类似于 master 数据库。本地连接到多租户数据库，sqlplus 连接上后，如何确认当前连接的是 CDB 还是 PDB，或者是连接的哪个 PDB，方式如下：

```
sqlplus / as sysdba

SQL*Plus: Release 12.1.0.2.0 Production on Wed Jul 15 13:26:54 2015

Copyright (c) 1982, 2014, Oracle. All rights reserved.

Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Production
With the Partitioning, Real Application Clusters, Automatic Storage Management,
Oracle Label Security,
OLAP, Advanced Analytics and Real Application Testing options

SQL> select name, open_mode from v$database;

NAME      OPEN_MODE
--------- --------------------
TEST   READ WRITE

SQL>
```

和传统的数据库使用 sqlplus 没有区别，但是知道这是一个容器数据库，那么如何分别登录到相应的 PDB 数据库呢？使用命令 show con_name 可以查看当前连接的是哪个数据库，以下显示当前连接为 CDB。

```
SQL> show con_name;

CON_NAME
------------------------------
CDB$ROOT
```

使用命令 alter session set container=PDB 名的方式能够登录到具体的某个 PDB 下：

```
SQL> show con_name;

CON_NAME
------------------------------
CDB$ROOT
```

当前 session 连接为 CDB：

```
SQL> select name from v$pdbs;--查看当前数据库里面有哪些 pdb

NAME
```

```
------------------------------
PDB$SEED
PDB TEST1
PDB TEST2

SQL> alter session set container=PDB_TEST1;--设置 session 连接到 PDB_TEST1

Session altered.

SQL> show con_name;

CON_NAME
------------------------------
PDB_TEST1
```

可以看到当前 session 已变为 PDB_TEST1。

连接到多租户数据库，使用客户端的方式同普通数据库没有区别，先看监听状态，当前数据库环境存在两个 PDB，分别为 PDB_TEST1、PDB_TEST2，一个 CDB 叫 test，可以看到监听中存在 3 个服务，分别为 PDB_TEST1、PDB_TEST2 和 test。

```
$lsnrctl status
LSNRCTL for Linux: Version 12.1.0.2.0 - Production on 15-JUL-2015 12:10:09

Copyright (c) 1991, 2014, Oracle. All rights reserved.

Connecting to (ADDRESS=(PROTOCOL=tcp)(HOST=)(PORT=1521))
STATUS of the LISTENER
------------------------
Alias                      LISTENER
Version                     TNSLSNR for Linux: Version 12.1.0.2.0 - Production
Start Date                  15-JUL-2015 10:13:04
Uptime                     0 days 1 hr. 57 min. 5 sec
Trace Level                 off
Security                   ON: Local OS Authentication
SNMP                       OFF
Listener Parameter File    /u01/app/12.1.0.2/grid/network/admin/listener.ora
Listener Log File         /u01/app/grid/diag/tnslsnr/node2/listener/alert/log.xml
Listening Endpoints Summary...
  (DESCRIPTION=(ADDRESS=(PROTOCOL=ipc)(KEY=LISTENER)))
  (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=192.168.188.102)(PORT=1521)))
  (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=192.168.188.104)(PORT=1521)))
Services Summary...
Service "+ASM" has 1 instance(s).
  Instance "+ASM2", status READY, has 1 handler(s) for this service...
Service "PDB_TEST2" has 1 instance(s).
  Instance "TEST 1", status READY, has 1 handler(s) for this service...
Service "PDB TEST1" has 1 instance(s).
  Instance "TEST_1", status READY, has 1 handler(s) for this service...
Service "TEST" has 1 instance(s).
  Instance "TEST_1", status READY, has 1 handler(s) for this service...
```

```
Service "testXDB" has 1 instance(s).
  Instance "TEST_1", status READY, has 1 handler(s) for this service...
```

下面采用简易连接的方式来连接数据库：

```
sqlplus system/Oracle@192.168.188.104:1521/test
SQL*Plus: Release 12.1.0.2.0 Production on Wed Jul 15 12:59:52 2015
Copyright (c) 1982, 2014, Oracle. All rights reserved.
Last Successful login time: Mon Jul 13 2015 13:05:52 +08:00
Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Production
With the Partitioning, Real Application Clusters, Automatic Storage Management,
Oracle Label Security,
OLAP, Advanced Analytics and Real Application Testing options
SQL> show con_name;
CON_NAME
------------------------------
CDB$ROOT
--连接 PDB TEST2
sqlplus system/Oracle@192.168.188.104:1521/PDB TEST2
SQL*Plus: Release 12.1.0.2.0 Production on Wed Jul 15 13:01:55 2015
Copyright (c) 1982, 2014, Oracle. All rights reserved.
Last Successful login time: Wed Jul 15 2015 12:59:52 +08:00
Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Production
With the Partitioning, Real Application Clusters, Automatic Storage Management,
OLAP,
Advanced Analytics and Real Application Testing options
SQL> show con_name;
CON NAME
------------------------------
PDB_TEST2
```

以上可以看到，容器数据库的连接方式和普通数据库的连接方式没有区别，同理，配置 tnsnames.ora 别名的方式也同普通数据库一样。

在可拔插容器数据库中，由于有 PDB 的存在，在启动或者停止过程中会有额外的操作，正常启动数据库如下：

```
sqlplus / as sysdba

SQL*Plus: Release 12.1.0.2.0 Production on Wed Jul 15 13:13:41 2015

Copyright (c) 1982, 2014, Oracle. All rights reserved.

Connected to an idle instance.

SQL> startup
ORACLE instance started.

Total System Global Area 1073741824 bytes
Fixed Size                  2932632 bytes
```

```
Variable Size            801112168 bytes
Database Buffers          264241152 bytes
Redo Buffers               5455872 bytes
Database mounted.
Database opened.
SQL> show con_name;

CON NAME
------------------------------
CDB$ROOT
```

可以看到已启动数据库，但是当前连接的是 CDB，可以看下 CDB 中的 PDB 状态：

```
SQL> select name，open mode from v$pdbs;

NAME                           OPEN_MODE
------------------------------ ----------
PDB$SEED                        READ ONLY
PDB TEST1                          MOUNTED
PDB_TEST2                          MOUNTED
```

可以看到数据库中的两个 PDB 状态均为 mounted，此时应用端用户是无法连接到各自相应的 PDB 数据库的。需要使 PDB 可读写还需执行如下操作：

```
SQL> alter pluggable database PDB_TEST1 open;

Pluggable Database altered.

SQL> select name,open_mode from v$pdbs;

NAME                           OPEN_MODE
------------------------------ ----------
PDB$SEED                        READ ONLY
PDB_TEST1                          READ WRITE
PDB_TEST2                          MOUNTED
```

现在 PDB_TEST1 已经为 read write 状态，如果需要关闭 PDB_TEST1，可以执行：

```
SQL> alter pluggable database PDB_TEST1 close;

Pluggable Database altered.

SQL> select name,open mode from v$pdbs;

NAME                           OPEN_MODE
------------------------------ ----------
PDB$SEED                        READ ONLY
PDB TEST1                          MOUNTED
PDB_TEST2                          MOUNTED
```

上面的命令是单独打开或者关闭某个特定的 PDB，还可以通过以下命令一次打开或者关闭所有的 PDB：

```
SQL>alter pluggable Database all open;
SQL>alter pluggable Database all close;
```

同时，还可以打开 PDB 数据库类似传统的方式，演示如下：

```
SQL> alter session set container=PDB_TEST1;

Session altered.

SQL> show con_name;

CON_NAME
------------------------------
PDB_TEST1
SQL> startup
Pluggable Database opened.
SQL> select name，open_mode from v$pdbs;

NAME                          OPEN_MODE
------------------------------ ----------
PDB_TEST1                         READ WRITE
```

9.3 小结

通过本章的学习，读者可以了解 Oracle 12c 版本多租户功能的特点和应用场景，该功能可以将大量数据库进行整合，在后端形成一套可以快速部署消费的数据库资源池，这也符合了云计算的概念，其实 Oracle 数据库在 12c 版本中还提供了其他方面的新特性，但多租户功能是最具代表性的，也是目前企业环境中使用相对比较广泛的功能模块。

从下一章节开始，将介绍 SQL 语句的使用方法，这部分的内容，无论是开发人员还是数据库管理员，都是最基本的知识点，是用户与数据库之间交互的语言。

第 10 章 SQL基础

SQL（结构化查询语言，Structured Query Language）是一种数据库查询和程序设计语言，用于存取数据以及查询、更新、管理关系数据库系统，同时也是数据库脚本文件的扩展名。SQL 语句无论是种类还是数量都是繁多的，很多语句也经常用到，SQL 查询语句 SELECT 就是一个典型的例子，无论是高级查询还是低级查询，SQL 查询语句的需求都是最频繁的。本章将为读者讲解各种不同的 SQL 语句。

10.1 SQL 语句分类

SQL 语句根据操作对象的不同可分为两大类型：数据操作语言（DML）和数据定义语言（DDL）。

DML（data manipulation language）包括 SELECT、UPDATE、INSERT、DELETE，就像它的名字一样，这 4 条命令是用来对数据库里的数据进行操作的语言。DDL（data definition language）主要的命令有 CREATE、ALTER、DROP 等，用在定义或改变表（TABLE）的结构、数据类型、表之间的链接和约束等初始化工作上，大多在建立表时使用。从使用频率来讲，DML 是日常经常使用的 SQL 类型，DDL 相对较少。

举例来看，常用的 DDL 语句包括：

- CREATE：创建数据库对象。
- ALTER：修改数据库或数据库对象的结构，比如修改数据库参数或修改数据表单。
- DROP：删除数据库对象。
- TRUNCATE：清空表单数据但保留表结构。
- GRANT：给数据库用户授权。
- REVOKE：收回已经授予的权限。
- SELECT：查询数据。
- INSERT：添加数据，插入新数据。
- UPDATE：对已有数据进行更新修改。

- DELETE：删除某行数据。

下面将对 SQL 语句的具体使用做详细介绍。

10.2 查询语句

10.2.1　SELECT 查询语句

SELECT 语句用于从表中选取数据，然后将结果存储在一个结果表中（称为结果集）。SELECT 语法如下：

```
select 列名称 from 表名称
```

或

```
select * from 表名称
```



SQL 语句对大小写不敏感。SELECT 等效于 select。

【示例 10-1】从名为 Persons 的数据库表（见表 10-1）中获取名为 LastName 和 FirstName 列的内容

```
select lastname,firstname from persons
```

结果如表 10-2 所示。

表 10-1　Persons表

Id	LastName	FirstName	Address	City
1	Adams	John	Oxford Street	London
2	Bush	George	Fifth Avenue	New York
3	Carter	Thomas	Changan Street	Beijing

表 10-2　结果

LastName	FirstName
Adams	John
Bush	George
Carter	Thomas

【示例 10-2】获取 Persons 表中所有的列（使用符号*取代列的名称）

```
select * from persons
```

结果如表 10-3 所示。

表 10-3　结果

Id	LastName	FirstName	Address	City
1	Adams	John	Oxford Street	London
2	Bush	George	Fifth Avenue	New York
3	Carter	Thomas	Changan Street	Beijing

10.2.2　SELECT DISTINCT 语句

很多时候，数据表中的数据可能会包含重复值。有时也许希望仅仅列出不重复（distinct）的值。这时可以使用关键词 DISTINCT 用于返回唯一不同的值。

语法：

```
select distinct 列名称 from 表名称
```

【示例 10-3】Order 表（见表 10-4）中选取 Company 列的所有值

```
select company from orders
```

表 10-4　Order表

Company	OrderNumber
IBM	3532
School	2356
Apple	4698
School	6953

结果如表 10-5 所示。

表 10-5　结果

Company
IBM
School
Apple

School



在结果集中，School 被列出了两次。

【示例 10-4】从 Company 中列出不重复的值（使用 SELECT DISTINCT 语句）

```
select distinct company from orders
```

结果如表 10-6 所示。

表 10-6　结果

Company
IBM
School
Apple

10.2.3　WHERE 子句

WHERE 子句用于规定选择的标准。如需有条件地从表中选取数据，可将 WHERE 子句添加到 SELECT 语句中，语法如下：

```
select 列名称 from 表名称 where 列 运算符 值
```

表 10-7 的运算符可在 WHERE 子句中使用。

表 10-7　WHERE 子句的使用方法

操作符	描述
=	等于
<>	不等于
>	大于
<	小于
>=	大于等于
<=	小于等于
BETWEEN	在某个范围内
LIKE	搜索某种模式



在某些版本的 SQL 中，操作符<>可以写为!=。

【示例 10-5】如果只希望选取表 10-8 中居住城市在 Beijing 的人，需要向 SELECT 语句添加 WHERE 子句：

```
select * from persons where city='beijing'
```

表 10-8　Persons 表

LastName	FirstName	Address	City	Year
Adams	John	Oxford Street	London	1970
Bush	George	Fifth Avenue	New York	1975
Carter	Thomas	Changan Street	Beijing	1980
Gates	Bill	Xuanwumen 10	Beijing	1985

结果参见表 10-9。

表 10-9　结果

LastName	FirstName	Address	City	Year
Carter	Thomas	Changan Street	Beijing	1980
Gates	Bill	Xuanwumen 10	Beijing	1985



在例子中的条件值周围使用的是单引号。SQL 使用单引号来环绕文本值（大部分数据库系统也接受双引号）。如果是数值，请不要使用引号。

如果条件列的内容是文本值，则下面第一条语句是正确的，第二条语句是错误的：

```
select * from persons where firstname='bush'
select * from persons where firstname=bush
```

如果条件列的内容是数值，则下面第一条语句是正确的，第二条语句是错误的：

```
select * from persons where year>1965
select * from persons where year>'1965'
```

10.2.4　AND 和 OR 运算符

AND 和 OR 可在 WHERE 子句中把两个或多个条件结合起来。如果第一个条件和第二个条件都成立，则 AND 运算符显示一条记录。如果第一个条件和第二个条件中只有一个成立，则 OR 运算符显示一条记录。本小节的示例数据参见表 10-10。

表 10-10　Person 表

LastName	FirstName	Address	City
Adams	John	Oxford Street	London
Bush	George	Fifth Avenue	New York
Carter	Thomas	Changan Street	Beijing
Carter	William	Xuanwumen 10	Beijing

【示例 10-6】使用 AND 来显示所有姓为 Carter 并且名为 Thomas 的人

```
select * from persons where firstname='thomas' and lastname='carter'
```

结果参见表 10-11。

表 10-11　结果

LastName	FirstName	Address	City
Carter	Thomas	Changan Street	Beijing

【示例 10-7】使用 OR 来显示所有姓为 Carter 或者名为 Thomas 的人

```
select * from persons where firstname='thomas' or lastname='carter'
```

结果参见表 10-12。

表 10-12　结果

LastName	FirstName	Address	City
Carter	Thomas	Changan Street	Beijing
Carter	William	Xuanwumen 10	Beijing

【示例 10-8】也可以把 AND 和 OR 结合起来（使用圆括号来组成复杂的表达式）

```
select * from persons where (firstname='thomas' or firstname='william') and
lastname='carter'
```

结果参见表 10-13。

表 10-13　结果

LastName	FirstName	Address	City
Carter	Thomas	Changan Street	Beijing
Carter	William	Xuanwumen 10	Beijing

10.2.5 ORDER BY 语句用于对结果集进行排序

ORDER BY 语句用于根据指定的列对结果集进行排序。语句默认按照升序对记录进行排序。如果希望按照降序对记录进行排序，可以使用 DESC 关键字。本小节演示数据参见表 10-14。

表 10-14 Orders 表

Company	OrderNumber
IBM	3532
School	2356
Apple	4698
School	6953

【示例 10-9】以字母顺序显示公司名称

```
select company,ordernumber from orders order by company
```

结果参见表 10-15。

表 10-15 结果

Company	OrderNumber
Apple	4698
IBM	3532
School	6953
School	2356

【示例 10-10】以字母顺序显示公司名称（Company），并以数字顺序显示顺序号（OrderNumber）

```
select company, ordernumber from orders order by company,ordernumber
```

结果参见表 10-16。

表 10-16 结果

Company	OrderNumber
Apple	4698
IBM	3532
School	2356
School	6953

【示例 10-11】以逆字母顺序显示公司名称

```
select company,ordernumber from orders order by company desc
```

结果参见表 10-17。

表 10-17　结果

Company	OrderNumber
School	6953
School	2356
IBM	3532
Apple	4698

【示例 10-12】以逆字母顺序显示公司名称并以数字顺序显示顺序号

```
select company, ordernumber from orders order by company desc, ordernumber asc
```

结果参见表 10-18。

表 10-18　结果

Company	OrderNumber
School	2356
School	6953
IBM	3532
Apple	4698



在以上的结果中有两个相等的公司名称（School）。只有这一次，在第一列中有相同的值时，第二列是以升序排列的。第一列中有些值为 nulls 时，情况也是这样的。

10.2.6　BETWEEN 操作符

操作符 BETWEEN AND 会选取介于两个值之间的数据范围。这些值可以是数值、文本或者日期。其语法如下：

```
select column_name(s)
from table_name
where column name
between value1 and value2
```

本小节数据参见表 10-19。

表 10-19 Persons

Id	LastName	FirstName	Address	City
1	Adams	John	Oxford Street	London
2	Bush	George	Fifth Avenue	New York
3	Carter	Thomas	Changan Street	Beijing
4	Gates	Bill	Xuanwumen 10	Beijing

【示例 10-13】以字母顺序显示介于 Adams（包括）和 Carter（不包括）之间的人

```
select * from persons
where lastname
between 'adams' and 'carter'
```

结果集如表 10-20 所示。

表 10-20 结果

Id	LastName	FirstName	Address	City
1	Adams	John	Oxford Street	London
2	Bush	George	Fifth Avenue	New York



不同的数据库对 BETWEEN...AND 操作符的处理方式是有差异的。某些数据库会列出介于 Adams 和 Carter 之间的人，但不包括 Adams 和 Carter；某些数据库会列出介于 Adams 和 Carter 之间并包括 Adams 和 Carter 的人；而另一些数据库会列出介于 Adams 和 Carter 之间的人，包括 Adams，但不包括 Carter。所以，请检查你的数据库是如何处理 BETWEEN..AND 操作符的！

【示例 10-14】使用 NOT 操作符显示【示例 10-13】范围之外的人

```
select * from persons
where lastname not between 'adams' and 'carter'
```

结果集参见表 10-21。

表 10-21 结果

Id	LastName	FirstName	Address	City
3	Carter	Thomas	Changan Street	Beijing
4	Gates	Bill	Xuanwumen 10	Beijing

10.2.7　LIKE 操作符

LIKE 操作符用于在 WHERE 子句中搜索列中的指定模式。语法如下：

```
select column_name(s)
from table name
where column_name like pattern
```

本小节数据参考表 10-22。

表 10-22　Persons

Id	LastName	FirstName	Address	City
1	Adams	John	Oxford Street	London
2	Bush	George	Fifth Avenue	New York
3	Carter	Thomas	Changan Street	Beijing

【示例 10-15】从 Persons 表中选取居住在以 N 开头的城市里的人

```
select * from persons
where city like 'n%'
```



"%"可用于定义通配符（条件模式中缺少的字母）。

结果集参见表 10-23。

表 10-23　结果

Id	LastName	FirstName	Address	City
2	Bush	George	Fifth Avenue	New York

【示例 10-16】从 Persons 表中选取居住在以 g 结尾的城市里的人

```
select * from persons
where city like '%g'
```

结果集参见表 10-24。

表 10-24　结果

Id	LastName	FirstName	Address	City
3	Carter	Thomas	Changan Street	Beijing

【示例 10-17】从 Persons 表中选取居住在包含 lon 的城市里的人

```
select * from persons
where city like '%lon%'
```

结果集参见表 10-25。

表 10-25　结果

Id	LastName	FirstName	Address	City
1	Adams	John	Oxford Street	London

【示例 10-18】使用 NOT 关键字从 Persons 表中选取居住在不包含 lon 的城市里的人

```
select * from persons
where city not like '%lon%'
```

结果集参见表 10-26。

表 10-26　结果

Id	LastName	FirstName	Address	City
2	Bush	George	Fifth Avenue	New York
3	Carter	Thomas	Changan Street	Beijing

10.3 数据操作语句

10.3.1　INSERT INTO 语句

INSERT INTO 语句用于向表格中插入新的行数据。语法如下：

```
insert into 表名称 values (值 1,值 2,....)
```

也可以指定所要插入数据的列：

```
insert into table_name (列 1, 列 2,...) values (值 1, 值 2,....)
```

插入新行使用的数据表参见表 10-27。

表 10-27　Persons表

LastName	FirstName	Address	City
Carter	Thomas	Changan Street	Beijing

【示例 10-19】使用 SQL 语句向表中插入整行数据

```
insert into persons values ('Gates','Bill', 'Xuanwumen 10', 'Beijing')
```

结果参见表 10-28。

表 10-28　结果

LastName	FirstName	Address	City
Carter	Thomas	Changan Street	Beijing
Gates	Bill	Xuanwumen 10	Beijing

【示例 10-20】在指定的列中插入数据

```
insert into persons (lastname,address) values ('Wilson', 'Champs-Elysees')
```

结果参见表 10-29。

表 10-29　结果

LastName	FirstName	Address	City
Carter	Thomas	Changan Street	Beijing
Gates	Bill	Xuanwumen 10	Beijing
Wilson		Champs-Elysees	

10.3.2　UPDATE 语句

UPDATE 语句用于修改表中的数据。语法如下：

```
update 表名称 set 列名称 = 新值 where 列名称 = 某值
```

要使用的 Person 表演示数据参见表 10-30。

表 10-30　Person表

LastName	FirstName	Address	City
Gates	Bill	Xuanwumen 10	Beijing
Wilson		Champs-Elysees	

【示例 10-21】更新某一行中的一个列（为 lastname 是 Wilson 的人添加 firstname）

```
update person set firstname = 'Fred' where lastname = 'Wilson'
```

结果参见表 10-31。

表 10-31　结果

LastName	FirstName	Address	City
Gates	Bill	Xuanwumen 10	Beijing
Wilson	Fred	Champs-Elysees	

【示例 10-22】更新某一行中的若干列（修改地址（address）并添加城市名称（city））

```
update person set address = 'Zhongshan 23',  city = 'Nanjing'
where lastname = 'Wilson'
```

结果参见表 10-32。

表 10-32　结果

LastName	FirstName	Address	City
Gates	Bill	Xuanwumen 10	Beijing
Wilson	Fred	Zhongshan 23	Nanjing

10.3.3　DELETE 语句

DELETE 语句用于删除表中的行。语法非常简单，具体如下：

```
delete from 表名称 where 列名称 = 值
```

Person 表数据参见 10-33。

表 10-33　Person表

LastName	FirstName	Address	City
Gates	Bill	Xuanwumen 10	Beijing
Wilson	Fred	Zhongshan 23	Nanjing

【示例 10-23】删除表中某行数据（这里指定条件，Fred Wilson 会被删除）：

```
delete from person where lastname = 'wilson'
```

结果参见表 10-34。

表 10-34　结果

LastName	FirstName	Address	City
Gates	Bill	Xuanwumen 10	Beijing

【示例 10-24】删除所有行

可以在不删除表的情况下删除所有行。这意味着表的结构、属性和索引都是完整的：

```
delete from table_name
```

或者

```
delete * from table_name
```

10.4 连接查询语句

10.4.1　JOIN 和 KEY 的作用

有时为了得到完整的结果，需要从两个或更多的表中获取结果。就需要执行 JOIN。

数据库中的表可通过键将彼此联系起来。主键（Primary Key）是一个列，在这个列中的每一行的值都是唯一的。在表中，每个主键的值都是唯一的。这样做的目的是在不重复每个表中的所有数据的情况下，把表间的数据交叉捆绑在一起。

Persons 表数据参见表 10-35。

表 10-35　Persons表

Id_P	LastName	FirstName	Address	City
1	Adams	John	Oxford Street	London
2	Bush	George	Fifth Avenue	New York
3	Carter	Thomas	Changan Street	Beijing

注意，Id_P 列是 Persons 表中的主键。这意味着没有两行能够拥有相同的 Id_P。即使两个人的姓名完全相同，Id_P 也可以区分它们。

接下来请看 Orders 表，如表 10-36 所示。

表 10-36　Orders表

Id_O	OrderNo	Id_P
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	65



Id_O 列是 Orders 表中的主键，同时，Orders 表中的 Id_P 列用于引用 Persons 表中的人，而无须使用他们的确切姓名。请留意，Id_P 列把上面的两个表联系了起来。

【示例 10-25】通过引用两个表的方式从两个表中获取订购了产品的人以及订购的产品信息

```
select persons.lastname, persons.firstname,orders.orderno
from persons, orders
where persons.id_p = orders.id_p
```

结果集参见表 10-37。

表 10-37　结果

LastName	FirstName	OrderNo
Adams	John	22456
Adams	John	24562
Carter	Thomas	77895
Carter	Thomas	44678

除了上面的方法，也可以使用关键词 JOIN 来从两个表中获取数据。

【示例 10-26】列出所有人的订购信息

```
select persons.lastname， persons.firstname， orders.orderno
from persons
inner join orders
on persons.id_p = orders.id_p
order by persons.lastname
```

结果集参见表 10-38。

表 10-38　结果

LastName	FirstName	OrderNo
Adams	John	22456
Adams	John	24562
Carter	Thomas	77895
Carter	Thomas	44678

除了在上面的例子中使用的 INNER JOIN（内连接），还可以使用其他几种连接。下面列出可以使用的 JOIN 类型以及它们之间的差异。

- JOIN：如果表中有至少一个匹配，则返回行。
- LEFT JOIN：即使右表中没有匹配，也从左表返回所有的行。
- RIGHT JOIN：即使左表中没有匹配，也从右表返回所有的行。
- FULL JOIN：只要其中一个表中存在匹配，就返回行。

以上是主要的查询连接方式，下面将对每种连接方式的使用方式做逐一介绍。

10.4.2　INNER JOIN 关键字

当在表中存在至少一个匹配时，可以使用 INNER JOIN 关键字查询行数据，语法如下：

```
select column_name(s)
from table name1
inner join table name2
on table_name1.column_name=table_name2.column_name
```



INNER JOIN 与 JOIN 是相同的。

Persons 表如表 10-39 所示。

表 10-39　Persons表

Id_P	LastName	FirstName	Address	City
1	Adams	John	Oxford Street	London
2	Bush	George	Fifth Avenue	New York
3	Carter	Thomas	Changan Street	Beijing

Orders 表如表 10-40 所示。

表 10-40　Orders表

Id_O	OrderNo	Id_P
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	65

【示例 10-27】列出所有人的订购信息

```
select persons.lastname， persons.firstname,orders.orderno
from persons
inner join orders
on persons.id p=orders.id p
order by persons.lastname
```

结果集参见表 10-41。INNER JOIN 关键字在表中存在至少一个匹配时返回行。如果 Persons 中的行在 Orders 中没有匹配，就不会列出这些行。

表 10-41　结果

LastName	FirstName	OrderNo
Adams	John	22456
Adams	John	24562
Carter	Thomas	77895
Carter	Thomas	44678

10.4.3　LEFT JOIN 关键字

LEFT JOIN 关键字会从左表（table_name1）返回所有的行，即使在右表（table_name2） 中没有匹配的行。语法如下：

```
select column name(s)
from table_name1
left join table_name2
on table_name1.column_name=table_name2.column_name
```



在某些数据库中，LEFT JOIN 称为 LEFT OUTER JOIN。

还是上一小节的 Persons 表和 Orders 表。

【示例 10-28】列出所有的人以及他们的订购信息（如果有的话）

```
select persons.lastname,  persons.firstname,orders.orderno
from persons
left join orders
on persons.id_p=orders.id_p
order by persons.lastname
```

结果集参见表 10-42。LEFT JOIN 关键字会从左表（Persons）返回所有的行，即使在右表（Orders）中没有匹配的行。

表 10-42　结果

LastName	FirstName	OrderNo
Adams	John	22456
Adams	John	24562
Carter	Thomas	77895
Carter	Thomas	44678
Bush	George	

10.4.4　RIGHT JOIN 关键字

RIGHT JOIN 关键字会从右表（table_name2）返回所有的行，即使在左表 （table_name1）中没有匹配的行。语法如下：

```
select column_name(s)
from table_name1
right join table_name2
on table_name1.column_name=table_name2.column_name
```



在某些数据库中，RIGHT JOIN 称为 RIGHT OUTER JOIN。

还是上一小节的 Persons 表和 Orders 表。

【示例 10-29】列出所有的订单以及订购的人（如果有的话）

```
select persons.lastname, persons.firstname,orders.orderno
from persons
right join orders
on persons.id p=orders.id p
order by persons.lastname
```

结果集参见表 10-43。RIGHT JOIN 关键字会从右表（Orders）返回所有的行，即使在左表（Persons）中没有匹配的行。

表 10-43　结果

LastName	FirstName	OrderNo
Adams	John	22456
Adams	John	24562
Carter	Thomas	77895
Carter	Thomas	44678
		34764

10.4.5　FULL JOIN 关键字

FULL JOIN 又称全连接，只要其中某个表存在匹配，FULL JOIN 关键字就会返回行。语法如下：

```
select column name(s)
from table name1
full join table_name2
on table_name1.column_name=table_name2.column_name
```



在某些数据库中，FULL JOIN 称为 FULL OUTER JOIN。

还是上一小节的 Persons 表和 Orders 表。

【示例 10-30】列出所有的人及其订单，以及所有的订单和订购之人

```
select persons.lastname, persons.firstname, orders.orderno
from persons
full join orders
on persons.id_p=orders.id_p
order by persons.lastname
```

结果集参见表 10-44。FULL JOIN 关键字会从左表（Persons）和右表（Orders）返回所有的行。如果 Persons 中的行在表 Orders 中没有匹配，或者 Orders 中的行在表 Persons 中没有匹配，这些行同样会列出。

表 10-44 结果

LastName	FirstName	OrderNo
Adams	John	22456
Adams	John	24562
Carter	Thomas	77895
Carter	Thomas	44678
Bush	George	
		34764

10.4.6 UNION 操作符

UNION 操作符用于合并两个或多个 SELECT 语句的结果集。UNION 内部的 SELECT 语句必须拥有相同数量的列。列也必须拥有相似的数据类型。同时，每条 SELECT 语句中的列的顺序必须相同。UNION 语法如下：

```
select column_name(s) from table_name1
union
select column_name(s) from table_name2
```

默认地，UNION 操作符选取不同的值。如果允许重复的值，请使用 UNION ALL。UNION ALL 语法如下：

```
select column_name(s) from table_name1
union all
select column_name(s) from table_name2
```

另外，UNION 结果集中的列名总是等于 UNION 中第一个 SELECT 语句中的列名。

Employees_China 表如表 10-45 所示。

表 10-45　Employees_China表

E_ID	E_Name
01	Zhang，Hua
02	Wang，Wei
03	Carter，Thomas
04	Yang，Ming

Employees_USA 表如表 10-46 所示。

表 10-46　Employees_USA 表

E_ID	E_Name
01	Adams，John
02	Bush，George
03	Carter，Thomas
04	Gates，Bill

【示例 10-31】列出所有在中国和美国的不同的雇员名

```
select e name from employees china
union
select e_name from employees_usa
```

结果参见表 10-47。

表 10-47　结果

E_Name
Zhang，　Hua
Wang，　Wei
Carter，　Thomas
Yang，　Ming
Adams，　John
Bush，　George
Gates，　Bill



这个命令无法列出在中国和美国的所有雇员。在上面的例子中，有两个名字相同的雇员，他们当中只有一个人被列出来了。UNION 命令只会选取不同的值。

UNION ALL 命令和 UNION 命令几乎是等效的，不过 UNION ALL 命令会列出所有的值。

【示例 10-32】列出在中国和美国的所有雇员

```
select e name from employees china
union all
select e_name from employees_usa
```

结果参见表 10-48。

表 10-48　结果

E_Name
Zhang，　Hua
Wang，　Wei
Carter，　Thomas
Yang，　Ming
Adams，　John
Bush，　George
Carter，　Thomas
Gates，　Bill

10.5 常见函数

本节内容假设已经拥有表 10-49 所示的 Orders 表。

表 10-49　Orders表

O_Id	OrderDate	OrderPrice	Customer
1	2008/12/29	1000	Bush
2	2008/11/23	1600	Carter
3	2008/10/05	700	Bush
4	2008/09/28	300	Bush
5	2008/08/06	2000	Adams
6	2008/07/21	100	Carter

10.5.1　COUNT()函数

COUNT(column_name) 函数返回指定列的值的数目（NULL 值不被计入），其语法如下：

```
select count(column_name) from table_name
```

COUNT(*)函数返回表中所有的记录数：

```
select count(*) from table_name
```

COUNT(DISTINCT column_name)函数返回指定列的不同值的数目：

```
select count(distinct column_name) from table_name
```

【示例 10-33】计算客户 Carter 的订单数

```
select count(customer) as customernilsen from orders
where customer='carter'
```

以上 SQL 语句的结果如下，因为客户 Carter 共有 2 个订单：

```
CustomerNilsen
2
```

【示例 10-34】省略 WHERE 子句

```
select count(*) as numberoforders from orders
```

结果集如下，这是表中的总行数：

```
NumberOfOrders
6
```

【示例 10-35】计算 Orders 表中不同客户的数目

```
select count(distinct customer) as numberofcustomers from orders
```

结果集如下，这是 Orders 表中不同客户（Bush、Carter 和 Adams）的数目。

```
NumberOfCustomers
3
```

10.5.2　SUM()函数

SUM 函数返回数值列的总数（总额），相当于相加的功能。语法如下：

```
select sum(column_name) from table_name
```

【示例 10-36】查找 OrderPrice 字段的总数

```
select sum(orderprice) as ordertotal from orders
```

结果集如下：

```
OrderTotal
5700
```

10.5.3　MAX()函数

MAX 函数返回一列中的最大值，NULL 值不包括在计算中。语法如下：

```
select max(column_name) from table_name
```



MIN 和 MAX 也可用于文本列，以获得按字母顺序排列的最高或最低值。

【示例 10-37】查找 OrderPrice 列的最大值

```
select max(orderprice) as largestorderprice from orders
```

结果集类似这样：

```
LargestOrderPrice
2000
```

10.5.4　MIN()函数

MIN 函数返回一列中的最小值，NULL 值不包括在计算中。语法如下：

```
select min(column_name) from table_name
```

【示例 10-38】查找 OrderPrice 列的最小值

```
select min(orderprice) as smallestorderprice from orders
```

结果集如下：

```
SmallestOrderPrice
100
```

10.5.5　GROUP BY 语句

GROUP BY 语句用于结合合计函数，根据一个或多个列对结果集进行分组。语法如下：

```
select column name, aggregate function(column name)
from table_name
where column_name operator value
group by column_name
```

【示例 10-39】查找每个客户的总金额（使用 GROUP BY 语句对客户进行组合）

```
select customer,sum(orderprice) from orders
group by customer
```

结果集参见表 10-50。

表 10-50　结果

Customer	SUM(OrderPrice)
Bush	2000
Carter	1700
Adams	2000

【示例 10-40】省略 GROUP BY 出现的情况

```
select customer，sum(orderprice) from orders
```

结果集参见表 10-51。

表 10-51　结果

Customer	SUM(OrderPrice)
Bush	5700
Carter	5700
Bush	5700
Bush	5700
Adams	5700
Carter	5700

上面的结果集不是我们所需要的。那么为什么不能使用上面这条 SELECT 语句呢？因为上面的 SELECT 语句指定了两列（Customer 和 SUM(OrderPrice)），SUM(OrderPrice)返回一个单独的值（OrderPrice 列的总计），而 Customer 返回 6 个值（每个值对应 Orders 表中的每一行），因此得不到正确的结果。不过，读者已经看到了，GROUP BY 语句解决了这个问题。

10.6 小结

本章学习了基本的 SQL 语句使用方法，并且以上所罗列的内容均是日常数据库管理使用频率最多的 SQL 语句，希望读者能够在个人测试环境多加练习。SQL 语句本身没有难点，关键在于使用的熟练程度。

第 11 章 PL/SQL基础

PL/SQL 是一种高级数据库程序设计语言，专门用于在各种环境下对 Oracle 数据库进行访问。由于该语言集成于数据库服务器中，因此 PL/SQL 代码可以对数据进行快速高效的处理。除此之外，可以在 Oracle 数据库的某些客户端工具中使用 PL/SQL 语言。PL/SQL 是 Oracle 对标准数据库语言的扩展，Oracle 公司已经将 PL/SQL 整合到 Oracle 服务器和其他工具中了，近几年来有更多的开发人员和 DBA 开始使用 PL/SQL。

本章将讲述 PL/SQL 基础语法、结构以及如何设计并执行一个 PL/SQL 程序。

11.1 PL/SQL 的优点

从 Oracle 的版本 6 开始 PL/SQL 就被整合到 Oracle 中了，一旦掌握 PL/SQL 的优点以及其独有的数据管理的便利性，就很难想象 Oracle 缺了 PL/SQL 的情形。PL/SQL 不是一个独立的产品，而是一个整合到 Oracle 服务器和 Oracle 工具中的技术，可以把 PL/SQL 看作 Oracle 服务器内的一个引擎。

SQL 语句执行者处理单个的 SQL 语句，PL/SQL 引擎处理 PL/SQL 程序块。当 PL/SQL 程序块在 PL/SQL 引擎处理时，Oracle 服务器中的 SQL 语句执行器处理 PL/SQL 程序块中的 SQL 语句。

PL/SQL 的优点：

- PL/SQL 是一种高性能的基于事务处理的语言，能运行在任何 Oracle 环境中，支持所有数据处理命令。通过使用 PL/SQL 程序单元处理 SQL 的数据定义和数据控制元素。
- PL/SQL 支持所有 SQL 数据类型和所有 SQL 函数，同时支持所有 Oracle 对象类型。
- PL/SQL 块可以被命名和存储在 Oracle 服务器中，同时也能被其他的 PL/SQL 程序或 SQL 命令调用，任何客户/服务器工具都能访问 PL/SQL 程序，具有很好的可重用性。
- 可以使用 Oracle 数据工具管理存储在服务器中的 PL/SQL 程序的安全性。可以授权或撤销数据库其他用户访问 PL/SQL 程序的能力。
- PL/SQL 代码可以使用任何 ASCII 文本编辑器编写，所以对任何 Oracle 能够运行的操作系统都是非常便利的。
- 对于 SQL，Oracle 必须在同一时间处理每一条 SQL 语句，在网络环境下这就意味着每一个独立的调用都必须被 Oracle 服务器处理，这就占用大量的服务器时间，同时导致网络拥挤。PL/SQL 是以整个语句块发给服务器，这就降低了网络拥挤。

11.2 PL/SQL 块结构

PL/SQL 是一种块结构的语言，组成 PL/SQL 程序的单元是逻辑块，一个 PL/SQL 程序包含了一个或多个逻辑块，每个块都可以划分为三个部分。与其他语言相同，变量在使用之前必须声明，PL/SQL 提供了独立的专门用于处理异常的部分。下面描述 PL/SQL 块的不同部分：

- 声明部分（declaration section）：声明部分包含了变量和常量的数据类型和初始值。这个部分是由关键字 DECLARE 开始的，如果不需要声明变量或常量，那么可以忽略这一部分。需要说明的是游标的声明也在这一部分。
- 执行部分(executable section)：执行部分是 PL/SQL 块中的指令部分，由关键字 BEGIN 开始，所有的可执行语句都放在这一部分，其他的 PL/SQL 块也可以放在这一部分。
- 异常处理部分（exception section）：可选，在这一部分中处理异常或错误，对异常处理的详细讨论我们在后面进行。

PL/SQL 块语法：

```
declare:声明部分
begin
编写主题
exception 捕获异常
end;
```

PL/SQL 块中的每一条语句都必须以分号结束，SQL 语句可以是多行的，但分号表示该语句的结束。一行中可以有多条 SQL 语句，它们之间以分号分隔。每一个 PL/SQL 块由 BEGIN 或 DECLARE 开始，以 END 结束，注释由“---”标示，例如下面的代码片段。

【示例 11-1】注释

```
BEGIN
      -- 此存储过程用来清除 400 天前的运行线、调度命令数据

      -- 计算当前时间
      CUR TIME:= SYSDATE;
      CUR DAY:= TRUNC(CUR TIME);
      MYHOUR:= TO CHAR(CUR TIME, 'HH24');

      ------------------------ 整理调度命令表 ------------------------
      -- 当前调度命令表保留 400 天数据
      CMDDAYS:= 400;
      PRENDTIME:=  CUR DAY + (TO NUMBER(MYHOUR)/24) - ((24/24) * CMDDAYS);

      --清除当前表调度命令中 400 天前的数据
      DELETE FROM XD CMD RESULT WHERE (CMDID, DDT) IN (SELECT ID, DDT FROM XD CMD
WHERE SENDTIME < PRENDTIME);
      DELETE FROM XD CMD WIRELESS RESULT WHERE (CMDID, DDT) IN (SELECT ID, DDT FROM
XD CMD WHERE SENDTIME < PRENDTIME);
      DELETE FROM XD CMD CTCS WHERE (CMDID, DDT) IN (SELECT ID, DDT FROM XD CMD
WHERE SENDTIME < PRENDTIME);
      DELETE FROM XD CMD WHERE SENDTIME < PRENDTIME;
      DELETE FROM XD CMD RECV WHERE SENDTIME < PRENDTIME;
      COMMIT;
```

以上代码不要求读者对全部结构理解，主要是关注注释的用法。在注释里面可以写任何内容，主要是方便阅读代码。

11.3 PL/SQL 块的命名和匿名

PL/SQL 程序块既可以是一个命名的程序块，也可以是一个匿名程序块。匿名程序块既可以用在服务器端，也可以用在客户端。

命名程序块可以出现在其他 PL/SQL 程序块的声明部分。这方面比较明显的是子程序，子程序既可以在执行部分引用，也可以在异常处理部分引用。

PL/SQL 程序块可独立编译并存储在数据库中，任何与数据库相连接的应用程序都可以访问这些存储的 PL/SQL 程序块。Oracle 提供了四种类型的可存储程序：

- 函数
- 过程
- 包
- 触发器

11.3.1 函数

函数是命名了的、存储在数据库中的 PL/SQL 程序块。函数接受零个或多个输入参数，有一个返回值，返回值的数据类型在创建函数时定义。定义函数的语法如下：

```
function name [{parameter[,parameter,...])] return datatypes is [local
declarations]
begin
execute statements [exception
exception handlers] end [name]
```

11.3.2 过程

存储过程是一个 PL/SQL 程序块，接受零个或多个参数作为输入（INPUT）或输出（OUTPUT），或既作输入又作输出（INOUT）。与函数不同，存储过程没有返回值，不能由 SQL 语句直接使用，只能通过 EXECUTE 命令或 PL/SQL 程序块内部调用。定义存储过程的语法如下：

```
procedure name [(parameter[,parameter,...])] is [local declarations]
begin
execute statements [exception
exception handlers ] end [name]
```

【示例 11-2】创建一个存储过程（名字是 remove_emp，拥有者用户是 hr）

```
CREATE PROCEDURE remove_emp (employee_id NUMBER) AS
```

```
   tot emps NUMBER;
   BEGIN
      DELETE FROM employees
      WHERE employees.employee id = remove emp.employee id;
   tot emps := tot emps - 1;
   END;
/
```

创建存储过程，自然用到 create 命令，后面需要对存储过程进行命名，之后需要让存储过程知道具体要做哪些事情。在本例中，就是 delete 操作，这部分包含在 begin 和 end 中间。

11.3.3　包

包（package）其实就是被组合在一起的相关对象的集合，当包中任何函数或存储过程被调用，包就被加载入内存中，包中的任何函数或存储过程的子程序访问速度将大大加快。

包由两个部分组成：规范和包主体（body）。规范描述变量、常量、游标和子程序，包主体完全定义子程序和游标。

下面列举一个具有 body 的 PL/SQL 包，这里主要是希望让读者可以对 package 有一个直观的印象。

【示例 11-3】具有 body 的 PL/SQL 包

```
CREATE PACKAGE emp bonus AS
   PROCEDURE calc_bonus (date_hired employees.hire_date%TYPE);
END emp bonus;
/
CREATE PACKAGE BODY emp bonus AS
-- the following parameter declaration raises an exception
-- because 'DATE' does not match employees.hire date%TYPE
-- PROCEDURE calc bonus (date hired DATE) IS
-- the following is correct because there is an exact match
   PROCEDURE calc bonus
     (date hired employees.hire date%TYPE) IS
   BEGIN
     DBMS_OUTPUT.PUT_LINE
       ('Employees hired on ' || date hired || ' get bonus.');
   END;
END emp bonus;
/
```

Package 的创建跟上面的存储过程类似，首先需要用到 create 创建关键字，而后需要对 Package 指定名字和对应的存储过程，如果存储过程事先没有创建，可以一并写在代码里面。

11.3.4　触发器

触发器（trigger）与一个表或数据库事件联系在一起的，当一个触发器事件发生时，定义在表上的触发器被触发。其结构如下：

```
create [or replace] tigger 触发器名 触发时间 触发事件
```

```
on 表名
[for each row]
begin
 pl/sql 语句
end
```

下面列举一个简单的触发器，帮助读者理解其功能和代码结构。

【示例 11-4】简单的触发器

```
create or replace trigger auth secure before insert or update or DELETE
on tb emp
begin
  IF(to char(sysdate,'DY')='星期日') THEN
    RAISE APPLICATION ERROR(-20600,'不能在周末修改表 tb emp');
  END IF;
END;
/
```

上面的触发器在更新表 tb_emp 之前触发，目的是不允许在周末修改表。

11.3.5　声明变量

变量存放在内存中以获得值，能被 PL/SQL 块引用。我们可以把变量想象成一个可储藏东西的容器，容器内的东西是可以改变的。变量一般都在 PL/SQL 块的声明部分声明，PL/SQL 是一种强壮的类型语言，这就是说在引用变量前必须首先声明，要在执行或异常处理部分使用变量，那么变量必须首先在声明部分进行声明。

声明变量的语法如下：

```
variable_name [constant] databyte [not null][:=|default expression]
```



可以在声明变量的同时给变量强制性地加上 NOT NULL 约束条件，此时变量在初始化时必须赋值。

11.3.6　给变量赋值

给变量赋值有两种方式：

- 直接给变量赋值

```
x:=200; y=y+(x*20);
```

- 通过 SQL SELECT INTO 或 FETCH INTO 给变量赋值

```
select sum(salary),sum(salary*0.1) into total salary,tatal commission from
employee
where dept=10;
```

11.3.7　常量

常量与变量相似，但常量的值在程序内部不能改变。常量的值在定义时赋予，声明方式与变量相似，但必须包括关键字 CONSTANT。常量和变量都可被定义为 SQL 和用户定义的数据类型。

```
zero_value constant number:=0;
```

这个语句定义了一个名叫 ZERO_VALUE、数据类型是 NUMBER、值为 0 的常量。

11.3.8　标量

标量（scalar）数据类型没有内部组件，它们大致可分为以下四类：

- number
- character
- date/time
- boolean

表 11-1 显示了数字数据类型，表 11-2 显示了字符数据类型，表 11-3 显示了日期和布尔数据类型。

表 11-1　数字数据类型

类型	Range	子类型	描述
BINARY_INTEGER	-214748~2147483647	NATURAL NATURAL NPOSITIVE POSITIVEN SIGNTYPE	用于存储单字节整数 要求存储长度低于 NUMBER 值，用于限制范围的子类型（SUBTYPE） ● NATURAL：用于非负数 ● POSITIVE：只用于正数 ● NATURALN：只用于非负数和非 NULL 值 ● POSITIVEN：只用于正数，不能用于 NULL 值 ● SIGNTYPE：只有值-1、0 或 1
NUMBER	1.0E-130~9.99E125	DEC DECIMAL DOUBLE PRECISION FLOAT INTEGERIC INT NUMERIC REAL SMALLINT	存储数字值，包括整数和浮点数。可以选择精度和刻度方式，语法： `number[（[,]）]` 默认的精度是 38，scale 是 0
PLS_INTEGER	-2147483647~2147483647		与 BINARY_INTEGER 基本相同，但采用机器运算时，PLS_INTEGER 会提供更好的性能

表 11-2　字符数据类型

类型	Rang	子类型	描述
CHAR	最大长度32767 字节	CHARACTER	存储定长字符串，如果长度没有确定，默认是 1
LONG	最大长度 2147483647 字节		存储可变长度字符串
RAW	最大长度32767 字节		用于存储二进制数据和字节字符串，当在两个数据库之间进行传递时，RAW 数据不在字符集之间进行转换
LONGRAW	最大长度 2147483647 字节		与LONG 数据类型相似，同样它也不能在字符集之间进行转换
ROWID	18 个字节		与数据库 ROWID 伪列类型相同，能够存储一个

表 11-3　DATE和BOOLEAN

类型	Range	描述
BOOLEAN	TRUE/FALSE	存储逻辑值 TRUE 或 FALSE，无参数
DATE	01/01/4712 BC	存储固定长的日期和时间值，日期值中包含时间

11.3.9　操作符

与其他程序设计语言相同，PL/SQL 有一系列操作符。操作符分为下面几类：

- 算术操作符
- 关系操作符
- 比较操作符
- 逻辑操作符

算术操作符的列举如表 11-4 所示。

表 11-4　算术操作符

操作符	描述
+	加
-	减
/	除
*	乘
**	乘方

关系操作符主要用于条件判断语句或用于 WHERE 子串中，关系操作符检查条件和结果是否为 TRUE 或 FALSE，如表 11-5 所示。

表 11-5 PL/SQL 中的关系操作符

操作符	描述
<	小于操作符
<=	小于或等于操作符
>	大于操作符
>=	大于或等于操作符
=	等于操作符
!=	不等于操作符
<>	不等于操作符
:=	赋值操作符

下面是比较操作符的列举，如表 11-6 所示。

表 11-6 比较操作符

操作符	描述
IS NULL	如果操作数为 NULL 就返回 TRUE
LIKE	比较字符串值
BETWEEN	验证值是否在范围之内
IN	验证操作数在设定的一系列值中

下面是逻辑操作符列举，见表 11-7。

表 11-7 是逻辑操作符

操作符	描述
AND	两个条件都必须满足
OR	只要满足两个条件中的一个
NOT	取反

11.3.10 执行部分

执行部分包含了所有的语句和表达式，执行部分以关键字 BEGIN 开始、以关键字 EXCEPTION 结束，如果 EXCEPTION 不存在，那么将以关键字 END 结束。分号分隔每一条语句，使用赋值操作符:=或 SELECT。

INTO 或 FETCH INTO 给每个变量赋值，执行部分的错误将在异常处理部分解决，在执行部分中可以使用另一个 PL/SQL 程序块，这种程序块被称为嵌套块。

所有的 SQL 数据操作语句都可以用于执行部分，PL/SQL 块不能在屏幕上显示 SELECT 语句的输出。

SELECT 语句必须包括一个 INTO 子串或者是游标的一部分。执行部分使用的变量和常量必须首先在声明部分声明。执行部分必须至少包括一条可执行语句。NULL 是一条合法的可执行语句，事务控制语句 COMMIT 和 ROLLBACK 可以在执行部分使用，数据定义语言（Data Definition language）不能在执行部分使用。DDL 语句与 execute immediate 一起使用或者是 DBMS_SQL 调用。

下面给出一个完整的数据比对示例。

【示例 11-5】数据比对

```
Declare
  vnum Number := 1;
  varBranchNo varchar2(8):=null;
  vartempSQL Varchar2(2000) :=null;
  vErrMsg   varchar2(2000) := null;

  CURSOR cursor_BranchNo Is  select distinct prov_branch_no from branch_def where
1=1 ;

Begin

    vErrMsg := '0';
    truncate table cust_relation_rep2_0127;

    Open cursor BranchNo;

    Loop
        FETCH cursor_BranchNo INTO varBranchNo;   -- 注释可以写在这里，说明该部分的作用
        Exit When cursor_BranchNo%Notfound;

        INSERT INTO clean data log 201802
VALUES(varBranchNo,'C2','CUST RELATION','0',SYSDATE,NULL, null,null);
        COMMIT;

        IF  varBranchNo <> '000002' THEN

           EXECUTE IMMEDIATE '
           INSERT /*+append parallel(32) */  INTO cust relation rep2 0127 nologging
           SELECT cust_id, branch_src, sys_src
           FROM cust_relation subpartition(p'||varBranchNo||'1)
           where cust_type=''P''
           GROUP BY cust_id,branch_src,sys_src
           HAVING COUNT(distinct cust no||branch src||sys src)>1
           ';
```

```
    EXECUTE IMMEDIATE '
    INSERT /*+append parallel(32) */  INTO cust relation rep2 0127 nologging
    SELECT cust_id, branch_src, sys_src
    FROM cust_relation subpartition(p'||varBranchNo||'2)
    where cust_type=''P''
    GROUP BY cust id,branch src,sys src
    HAVING COUNT(distinct cust no||branch src||sys src)>1
    ';

    EXECUTE IMMEDIATE '
    INSERT /*+append parallel(32) */  INTO cust_relation_rep2_0127 nologging
    SELECT cust id, branch src, sys src
    FROM cust relation subpartition(p'||varBranchNo||'3)
    where cust_type=''P''
    GROUP BY cust_id,branch_src,sys_src
    HAVING COUNT(distinct cust_no||branch_src||sys_src)>1
    ';

    EXECUTE IMMEDIATE '
    INSERT /*+append parallel(32) */  INTO cust_relation_rep2_0127 nologging
    SELECT cust_id, branch_src, sys_src
    FROM cust_relation subpartition(p'||varBranchNo||'4)
    where cust_type=''P''
    GROUP BY cust id,branch src,sys src
    HAVING COUNT(distinct cust no||branch src||sys src)>1
    ';

    EXECUTE IMMEDIATE '
    INSERT /*+append parallel(32) */  INTO cust_relation_rep2_0127 nologging
    SELECT cust id, branch src, sys src
    FROM cust relation subpartition(p'||varBranchNo||'5)
    where cust_type=''P''
    GROUP BY cust_id,branch_src,sys_src
    HAVING COUNT(distinct cust_no||branch_src||sys_src)>1
    ';

    EXECUTE IMMEDIATE '
    INSERT /*+append parallel(32) */  INTO cust_relation_rep2_0127 nologging
    SELECT cust_id, branch_src, sys_src
    FROM cust_relation subpartition(p'||varBranchNo||'6)
    where cust_type=''P''
    GROUP BY cust id,branch src,sys src
    HAVING COUNT(distinct cust no||branch src||sys src)>1
    ';
COMMIT;

END IF;

IF  varBranchNo = '000002' THEN
```

```
        EXECUTE IMMEDIATE '
         INSERT /*+append parallel(32) */  INTO cust_relation_rep2_0127 nologging
         SELECT cust id, branch src, sys src
         FROM cust relation partition(p'||varBranchNo||')
         where cust_type=''P''
         GROUP BY cust_id,branch_src,sys_src
         HAVING COUNT(distinct cust_no||branch_src||sys_src)>1
         ';


          commit;
      END IF;

      UPDATE clean data log 201802 SET end time=SYSDATE, task stat='2' WHERE
provbranch=varBranchNo AND clean type='C2';
      COMMIT;

    End Loop;
   CLOSE cursor_BranchNo;

End;
```

在代码的开头部分是对变量和游标的定义，之后在 begin 和 end 中间是程序真正执行的部分。

11.3.11 执行一个 PL/SQL 块

SQL*PLUS 中匿名的 PL/SQL 块的执行是在 PL/SQL 块后输入“/”来执行的，如下面的例子所示。

【示例 11-6】匿名 PL/SQL 块的执行

```
declare
v comm percent constant number:=10; begin
update emp
set comm=sal*v_comm_percent where deptno=10;
end SQL> /
PL/SQL procedure successfully completed.
```

命名的程序与匿名程序的执行不同，执行命名的程序块必须使用 EXECUTE 关键字。

【示例 11-7】执行命名的程序块

```
create or replace procedure update_commission
(v dept in number,v pervent in number default 10) is begin
update emp
set comm=sal*v_percent where deptno=v_dept;
end
SQL>/
Procedure created

SQL>execute update_commission(10,15);
```

```
PL/SQL procedure successfully completed.
SQL>
```

如果在另一个命名程序块或匿名程序块中执行这个程序，就不需要 EXECUTE 关键字了。

【示例 11-8】在其他程序块执行命名的程序

```
declare
v dept number; begin
select a.deptno into v_dept from emp a
where job='PRESIDENT' update_commission(v_dept);
end SQL>/
PL/SQL procedure successfully completed
SQL>
```

以上示例就是不需要使用 EXECUTE 关键字的情况。

11.3.12　控制结构

控制结构是 PL/SQL 程序的一种关于流程的代码形式，PL/SQL 支持条件控制和循环控制结构两种。

1. 条件控制结构

IF…THEN 语法如下：

```
if condition then statements 1;
statements 2;
.... end if
```

IF 语句判断条件 condition 是否为 TRUE，如果是，就执行 THEN 后面的语句，如果 condition 为 false 或 NULL 就跳过 THEN 到 END IF 之间的语句，执行 END IF 后面的语句。还有一种条件控制结构，即 IF…THEN…ELSE，语法如下：

```
if condition then statements 1;
statements 2;
.... else

statements 1;
statements 2;
.... end if
```

如果条件 condition 为 TRUE，则执行 THEN 到 ELSE 之间的语句，否则执行 ELSE 到 END IF 之间的语句。IF 可以嵌套，可以在 IF 或 IF…ELSE 语句中使用 IF 或 IF…ELSE 语句。

```
if (a>b) and (a>c) then g:=a;
else
g:=b;
if c>g then g:=c;
end if
end if
```

语法：

```
if condition1 then statement1;
elsif condition2 then statement2;
elsif condition3 then statement3;
else
statement4; end if;
statement5;
```

如果条件 condition1 为 TRUE 则执行 statement1，然后执行 statement5，否则判断 condition2 是否为 TRUE，若为 TRUE 则执行 statement2，然后执行 statement5，对于 condition3 也是相同的，如果 condition1、condition2、condition3 都不成立，那么将执行 statement4，然后执行 statement5。

2. 循环控制结构

循环控制的基本形式是 LOOP 语句，LOOP 和 ENDLOOP 之间的语句将无限次地执行。LOOP 语句的语法如下：

```
loop
statements;
end loop
```

LOOP 和 END LOOP 之间的语句无限次地执行显然是不行的，在使用 LOOP 语句时必须使用 EXIT 语句，强制循环结束，例如：

```
x:=100; loop
x:=x+10;
if x>1000 then exit;
end if end loop; y:=x;
```

此时 Y 的值是 1010，EXIT WHEN 语句将结束循环。如果条件为 TRUE，则结束循环 X:=100：

```
loop x:=x+10;
exit when x>1000; x:=x+10;
end loop; y:=x;

while..loop
```

WHILE..LOOP 有一个条件与循环相联系，如果条件为 TRUE，则执行循环体内的语句，如果结果为 FALSE，则结束循环。

```
x:=100;
while x<=1000 loop x:=x+10;
end loop; y=x;

for...loop
```

语法：

```
for counter in [reverse] start_range....end_range loop statements;
```

```
end loop;
```

LOOP 和 WHILE 循环的循环次数都是不确定的，FOR 循环的循环次数是固定的，counter 是一个隐式声明的变量，它的初始值是 start_range，第二个值是 start_range+1，直到 end_range，如果 start_range 等于 end_range，那么循环将执行一次。如果使用了 REVERSE 关键字，那么范围将是一个降序。

```
x:=100;
for v_counter in 1..10 loop x:=x+10;

end loop y:=x;
```

如果要退出 for 循环就可以使用 EXIT 语句。

11.4 实战 PL/SQL 举例

本节将举例说明 PL/SQL 的实际用法，主要向读者介绍几个比较常用和基础的 PL/SQL 用法，帮助读者理解其基本语法结构。

11.4.1 构造一个简单的 PL/SQL 块

执行以下 PL/SQL，构造一个简单的 PL/SQL 块。

【示例 11-9】构造 PL/SQL 块

```
declare
   i number;
begin
   i:=30;
   dbms output.put line('i 的内容为:'||i);
end;
```

此时，直接执行程序即可。执行之后发现没有任何的输出。因为 Oracle 在系统设置中默认设置了输出不显示，如果要显示的话，输入以下命令：

```
set serveroutput on;
```



输出字符串应该使用单引号。

11.4.2 PL/SQL 块接收用户的输入信息

例如，现在要求用户输入一个雇员编号，之后根据输入的内容进行查询，查询雇员的姓名，用户的输入信息使用“&”完成，首先构造一个测试表，后面的脚本操作会用到下面的数据。

【示例 11-10】构造测试表

```
CREATE TABLE EMP
      (EMPNO NUMBER(4) NOT NULL,
       ENAME VARCHAR2(10),
       JOB VARCHAR2(9),
       MGR NUMBER(4),
       HIREDATE DATE,
       SAL NUMBER(7, 2),
       COMM NUMBER(7, 2),
       DEPTNO NUMBER(2));

INSERT INTO EMP VALUES
       (7369, 'SMITH',  'CLERK',     7902,
       TO_DATE('17-11-1980', 'DD-MM-YYYY'),  800, NULL, 20);
INSERT INTO EMP VALUES
       (7499, 'ALLEN',  'SALESMAN', 7698,
       TO_DATE('20-11-1981', 'DD-MM-YYYY'), 1600,  300, 30);
INSERT INTO EMP VALUES
       (7521, 'WARD',   'SALESMAN', 7698,
       TO_DATE('22-11-1981', 'DD-MM-YYYY'), 1250,  500, 30);
INSERT INTO EMP VALUES
       (7566, 'JONES',  'MANAGER',  7839,
       TO_DATE('2-11-1981', 'DD-MM-YYYY'),  2975, NULL, 20);
INSERT INTO EMP VALUES
       (7654, 'MARTIN', 'SALESMAN', 7698,
       TO_DATE('28-11-1981', 'DD-MM-YYYY'), 1250, 1400, 30);
INSERT INTO EMP VALUES
       (7698, 'BLAKE',  'MANAGER',  7839,
       TO_DATE('1-11-1981', 'DD-MM-YYYY'),  2850, NULL, 30);
INSERT INTO EMP VALUES
       (7782, 'CLARK',  'MANAGER',  7839,
       TO_DATE('9-11-1981', 'DD-MM-YYYY'),  2450, NULL, 10);
INSERT INTO EMP VALUES
       (7788, 'SCOTT',  'ANALYST',  7566,
       TO_DATE('09-11-1982', 'DD-MM-YYYY'), 3000, NULL, 20);
INSERT INTO EMP VALUES
       (7839, 'KING',   'PRESIDENT', NULL,
       TO_DATE('17-11-1981', 'DD-MM-YYYY'), 5000, NULL, 10);
INSERT INTO EMP VALUES
       (7844, 'TURNER', 'SALESMAN', 7698,
       TO_DATE('8-11-1981', 'DD-MM-YYYY'),  1500,    0, 30);
INSERT INTO EMP VALUES
       (7876, 'ADAMS',  'CLERK',     7788,
       TO_DATE('12-11-1983', 'DD-MM-YYYY'), 1100, NULL, 20);
INSERT INTO EMP VALUES
       (7900, 'JAMES',  'CLERK',     7698,
       TO_DATE('3-10-1981', 'DD-MM-YYYY'),   950, NULL, 30);
INSERT INTO EMP VALUES
       (7902, 'FORD',   'ANALYST',  7566,
       TO_DATE('3-11-1981', 'DD-MM-YYYY'),  3000, NULL, 20);
```

```
INSERT INTO EMP VALUES
       (7934, 'MILLER', 'CLERK',     7782,
       TO DATE('23-12-1982', 'DD-MM-YYYY'), 1300, NULL, 10);

CREATE TABLE DEPT
      (DEPTNO NUMBER(2),
       DNAME VARCHAR2(14),
       LOC VARCHAR2(13) );

INSERT INTO DEPT VALUES (10, 'ACCOUNTING', 'NEW YORK');
INSERT INTO DEPT VALUES (20, 'RESEARCH',   'DALLAS');
INSERT INTO DEPT VALUES (30, 'SALES',      'CHICAGO');
INSERT INTO DEPT VALUES (40, 'OPERATIONS', 'BOSTON');
```

通过使用上面的代码，可以创建后面示例使用的测试表和测试数据。执行完毕之后，会创建名为 EMP 和 DEPT 的两个表，并且插入初始化测试数据。测试表创建完毕之后，进行如下测试。

【示例 11-11】测试

```
declare
   eno number;
   en varchar(20);
begin
   --输入的信息保存在 eno 里
   eno:=&no;
   --之后根据 eno 的值对数据库进行查询操作
   select ename into en from emp where empno = eno;
   dbms_output.put_line('编号为:'||eno||'雇员的姓名为:'||en);
exception
   when no_data_found then
   dbms output.put line('没有此雇员');
end;
```

执行以上代码时，系统会提示用户手工输入雇员的编号，当编号填写正确之后，如果数据库里面有记录，就会返回雇员的姓名，如果没有，返回的结果将是“没有此雇员”。上面的代码测试，dbms_output 包主要用于调试 PL/SQL 程序，或者在 sql*plus 命令中显示信息（displaying message）和报表，譬如我们可以写一个简单的匿名 PL/SQL 程序块，而该块出于某种目的使用 dbms_output 包来显示一些信息。

11.4.3　查询

在以上的查询中再进一步可以根据雇员的编号查出姓名及其领导的姓名和所在的部门进行显示。

【示例 11-12】查询信息

```
declare
   eno emp.empno%type ;
```

```
    en emp.ename%type ;
    mn emp.ename%type ;
    dn dept.dname%type ;
begin
    -- 输入的信息保存在 eno 里
    eno := &no ;
    -- 之后根据 eno 的值对数据库进行查询操作
    select  e.ename,m.ename,d.dname  into  en,mn,dn
    from  emp  e,dept  d,emp  m
    where e.empno=7369 and e.mgr=m.empno and e.deptno=d.deptno ;
   dbms_output.put_line('编号为：'||eno||'雇员的姓名为：'||en) ;
   dbms_output.put_line('编号为：'||eno||'雇员的上级姓名为：'||mn) ;
  dbms_output.put_line('编号为：'||eno||'雇员所在的部门：'||dn) ;
exception
    when no data found then
    dbms_output.put_line('没有此雇员') ;
end ;
```

说明：

- no_data_found 是一种异常类型：没有发现数据。
- emp.empno%TYPE 表示以 emp 表中的 empno 字段的类型定义变量。
- e.ename,m.ename,d.dname into en,mn,dn 表示一次可以同时放进去多个值。

PL/SQL 之中也包含了循环、分支等条件控制语句。以上代码执行之后，系统提示输入雇员编号，这里读者可以随便输入一个，显示结果如下：

```
编号为：30 雇员的姓名为：SMITH
编号为：30 雇员的上级姓名为：FORD
编号为：30 雇员所在的部门：RESEARCH
```

11.4.4 LOOP 循环

循环语句是我们在使用 Oracle 数据库时用的最多的语句之一，Oracle 中循环语句的写法很多，下面就让我们一起了解一下这些语句的写法。在 PL/SQL 中可以使用 LOOP 语句对数据进行循环处理，利用该语句可以循环执行指定的语句序列。其格式如下：

```
loop
    循环的语句  ;
exit when  终止条件  ;
循环条件必须更改  ;
end loop ;
```

例如，希望程序循环输出 1~10 的情况，可以使用下面的方法。

【示例 11-13】循环输出 1~10

```
declare
    countnum number ;
begin
```

```
    --必须赋初值
    countnum := 1 ;
    loop
        dbms_output.put_line('countnum = '||countnum) ;
    exit when countnum>10 ;
        countnum := countnum + 1 ;
    end loop ;
end ;
```

输出结果如下：

```
countnum = 1
countnum = 2
countnum = 3
countnum = 4
countnum = 5
countnum = 6
countnum = 7
countnum = 8
countnum = 9
countnum = 10
countnum = 11

PL/SQL 过程已成功完成。
```

通过上面的测试，达到了循环显示 1~10 的输出目的，关键是 LOOP 方法的使用，以 LOOP 开始、END LOOP 结束，中间包裹的部分就是需要循环执行的操作。



COUNT 关键字只能在 SQL 语句中使用，此循环是先执行一次之后再进行判断，执行结果到 11 结束循环。

11.4.5　WHILE 循环

WHILE 语句的历史更久，表达方式上更自由灵活，常用于无法事先判断循环次数的循环。

格式：

```
while(判断循环的条件)
loop 循环的语句  ;
    循环条件的改变  ;
end loop ;
```

【示例 11-14】使用 WHILE 语句修改前面的程序

```
declare
    countnum number ;
begin
    --必须赋初值
    countnum := 1 ;
    while(countnum<10)
```

```
    loop
       dbms_output.put_line('countnum = '||countnum) ;
       countnum := countnum + 1 ;
    end loop ;
end ;
```

此语句是先判断，如果满足条件则继续执行循环体，执行结果到 9 结束循环，输出结果如下：

```
countnum = 1
countnum = 2
countnum = 3
countnum = 4
countnum = 5
countnum = 6
countnum = 7
countnum = 8
countnum = 9

PL/SQL 过程已成功完成。
```

WHILE 循环也可以实现上一个例子的目的。但从代码的写法上看，WHILE 的作用是起到条件判断的，真正执行部分在 LOOP 和 END LOOP 之间。

11.4.6 FOR 循环

FOR 循环控制结构可以有效地编写需要执行的特定次数的循环，它与 WHILE 的区别是，WHILE 可以在不知道具体要循环多少次数的时候用，FOR 必须要知道循环多少次。FOR 循环格式如下：

```
for  变量名称  in  变量的初始值..结束值
    loop
    循环语句  ;
    end loop ;
```

【示例 11-15】修改【示例 11-14】

```
declare
    countnum number ;
begin
    for countnum in 1..10
    loop
       dbms output.put line('countnum = '||countnum) ;
    end loop ;
end ;
```

同 WHILE 类似，在上面的代码中，在 FOR 语句的后面会对要循环的范围进行一个判断，FOR 首先判断范围是 1~10，通过这行代码控制后面执行操作循环的次数。

此语句 countnum 大于等于 1 小于等于 10，最后输出 1~10，执行结果如下：

```
countnum = 1
countnum = 2
```

```
countnum = 3
countnum = 4
countnum = 5
countnum = 6
countnum = 7
countnum = 8
countnum = 9
countnum = 10

PL/SQL 过程已成功完成。
```



FOR 和 WHILE 都是先判断后执行，而 LOOP 是先执行后判断。FOR 循环和 WHILE 循环的区别是，FOR LOOP 中控制变量的初始化、条件判断和变量递增基本写在 FOR 后面的括号里，而 WHILE 都写在循环程序段里。

11.4.7　IF 语句

从单词的角度来理解，IF 就是条件判断语句，是比较基本的一个常见判断方法，格式如下：

```
if  条件  then
   满足条件时，执行此语句
end if;
```

下面举例进行测试：

【示例 11-16】IF 语句

```
declare
   countnum number ;
begin
   countnum := 11 ;
   if countnum>10 then
      dbms output.put line('countnum = '||countnum) ;
   end if ;
end ;
```

通过条件语句，判断当满足条件时执行一次，首先声明变量 countnum，之后对变量赋值，如果变量 countnum 大于 10，会执行后面的操作，输出数字“11”执行结果：

```
countnum = 11
```

11.4.8　IF…ELSE 语句

如果 IF 满足了，则执行 IF 语句块的内容，否则执行 ELSE。

【示例 11-17】IF…ELSE 语句

```
declare
   countnum number ;
begin
```

```
    countnum := 1 ;
    if countnum>10 then
        dbms output.put line('countnum = '||countnum) ;
    else
        dbms_output.put_line('条件不成立') ;
    end if ;
end ;
```

当在语句中 IF 后面的判断条件不满足时，执行 ELSE 后面的语句，执行结果如下：

```
条件不成立
```

11.4.9 IF…ELSIF…ELSE 语句

在 IF…ELSE 的基础上进一步扩展，例如：

```
declare
    countnum number ;
begin
    countnum := 1 ;
    if countnum>10 then
        dbms_output.put_line('countnum = '||countnum);
    elsif countnum<5 then
        dbms_output.put_line('值小于 5');
    else
        dbms output.put line('条件不满足');
    end if ;
end ;
```

上面的示例是一个多重判断语句方法，判断和返回结果由多个判断关键字控制，首先 if 判断变量 countnum 是否大于 10，如果大于，输入变量的值，如果小于 5，输出汉字“值小于 5”，当前面两个判断都不满足的时候，输出结果为“条件不满足”。

```
值小于 5
```

11.5 小结

本章介绍了 PL/SQL 的基础语法以及如何使用 PL/SQL 语言设计和运行 PL/SQL 程序块，并将 PL/SQL 程序整合到 Oracle 服务器中，虽然 PL/SQL 程序作为功能块嵌入 Oracle 数据库中，但 PL/SQL 与 Oracle 数据库的紧密结合使得越来越多的 Oracle 数据库管理员和开发人员开始使用 PL/SQL。

第 12 章

Oracle RAC架构安装与部署

本章向读者介绍 Oracle 数据库的高级安装案例，从一般了解学习的角度来说，安装单节点的 Oracle 数据库软件即可满足目的，但是对于企业生产环境来说，多数情况下，为了保障系统的稳定运行，都会避免存在单点故障的风险。所以，基本上在大型数据中心中，安装 Oracle RAC 架构已成为标配。

要了解 Oracle RAC 首先需要知道什么是 Cluster。一个 Cluster 是由两个或者多个独立的、通过网络连接的 servers 组成的。几个硬件供应商多年以来提供了 Cluster 性能的各种需求。一些 Cluster 仅仅为了提供高可用性，在当前活动的 node 发生故障时转移到次节点 node。另一些是为了提供分布式的连接、工作的可扩展性。Cluster 的共同特点是，对于一个应用程序，它可以看作一个单独的 server。同样，管理几个 server 应该尽可能像管理一个 server 一样简单。Cluster 管理器软件提供了这种功能。如果是 single server 的 node，文件必须存储在其各自 node 能访问的位置。存在有几个不同拓扑结构来解决数据访问的问题，这主要依赖于 Cluster 设计的主要目标。相互连接时一个物理的网络连接作为每个 Cluster 节点直接的交互通信。简而言之，一个 Cluster 就是一组独立的 server，它们共同协作，组成一个 single system。

RAC 是一个可以使你通过运行多个依赖相同 Database 的 Instance，使用 Cluster 硬件。数据库 files 被存放在物理或者逻辑上连接每个节点的磁盘上，以便于每个活动的 Instance 都可以对 files 进行读写操作。RAC 软件管理着数据的访问，所以更改操作在 Instances 之间是被相互协调的，并且每个 Instance 看到的信息和数据镜像都是一致的。通过 RAC 结构，可以获得冗余，从而使得即使在一个系统 crash 或者不可访问时，应用程序也可通过其他 Instance 访问 Database。

RAC 自动提供了服务的工作量管理。应用程序的服务可以被分组或分类，组成商业组件完成应用工作任务。RAC 中的服务可以是持续的、不间断的 Database 操作，并为多 Instance 上的多个服务提供支持。可以设计 services 到一个或多个 Instance 上运行，并且交替 Instance 可以用于备份 Instance。如果主 Instance 失败，Oracle 会将 service 从失败的 Instance 节点移动到活动的可替代的 Instance 上。Oracle 也会自动通过连接进行数据装载的平衡。RAC 利用多个廉价的 computer 共同提供 Database 的服务，就像一个大的 computer 一样，服务于只有大规模 SMP 才能提供的各种应用。

本章介绍大型企业中标准的 Oracle RAC 安装过程。

12.1 软件和硬件准备

建议按照下面的软件和硬件的基本要求进行安装前的确认工作。

- 服务器具有相同的体系架构（如 64bit 和 32bit 不能同一集群）。
- 非 cluster 环境要求物理内存高于 2GB，cluster 环境要求物理内存高于 8GB。

操作系统要求：

- 服务器运行相同版本的操作系统。
- 服务器操作系统运行在 3 或 5 的模式下（Linux）。
- 交换分区为物理内存的倍数（1~2GB SWAP=1.5RAM；2~16GB SWAP=RAM；大于 16GB SWAP=16GB）。
- TMP>1GB，磁盘空间>10GB。
- 分辨率要求至少 1024 × 768。
- 数据库版本：12.2.0.1。
- 节点数：2 节点。
- 操作系统版本：Redhat 7.4。

12.2 安装前的检查和配置

12.2.1 检查操作系统环境

检查操作系统体系结构：

```
uname -a
```

检查操作系统供应商及版本：

```
cat /etc/issue
```

检查物理内存：

```
cat /proc/meminfo |grep MemTotal
```

检查/dev/shm/大小（如果使用 Oracle 12c 的 AMM 内存管理方式， MEMORY_TARGET 和 MEMORY_MAX_TARGET 参数不能超过/dev/shm/大小）：

```
df -h /dev/shm/
```

调整大小可使用如下命令：

```
mount -o remount,size=7G /dev/shm
```

检查磁盘空间（安装 Oracle 软件的挂载点和/tmp）：

```
df -h
df -h /tmp
```

12.2.2　检查系统软件套件

对于 Linux 操作系统来说，在安装 Oracle 数据库软件之前，系统需要提前安装许多软件包，请读者按照如下的脚本进行安装前的检查。如果发现某些软件包没有提前安装，需要手动安装。

```
rpm -q --qf '%{NAME}-%{VERSION}-%{RELEASE} (%{ARCH})\n' binutils \
compat-libcap1 \
compat-libstdc++-33 \
e2fsprogs \
e2fsprogs-libs \
elfutils-libelf \
elfutils-libelf-devel \
gcc \
gcc-c++ \
glibc \
glibc-devel \
libaio \
libaio-devel \
libgcc \
libstdc++ \
libstdc++-devel \
make \
sysstat \
unixODBC \
ksh \
libX11 \
libXau \
libXi \
libXtst \
libxcb \
smartmontools \
unixODBC-devel \
net-tools |grep installed
```

经过上面的检查确认之后，如果全部的软件包或者个别软件包缺失可以有针对性地安装。下面的脚本示例是将所需的软件包全部安装的过程。

```
yum install -y binutils
yum install -y compat-libcap1
yum install -y compat-libstdc++-33
yum install -y e2fsprogs
yum install -y e2fsprogs-libs
yum install -y glibc
```

```
yum install -y glibc-devel
yum install -y ksh
yum install -y libgcc
yum install -y libstdc++
yum install -y libstdc++
yum install -y libaio
yum install -y libaio-devel
yum install -y libXtst
yum install -y libX11
yum install -y libXau
yum install -y libxcb
yum install -y libXi
yum install -y make
yum install -y net-tools
yum install -y sysstat
yum install -y smartmontools
yum install -y gcc-4.4.7
yum install -y gcc-c++
```

12.2.3 关闭服务（防火墙）

一般生产环境内部建议关闭操作系统的防火墙设置，因为多数情况下，网络的安全策略是由网络设备来接管的。关闭操作系统层面的安全设置，以避免更多的问题发生，也是为了方便数据库运行的最佳实践。

```
# 关闭防火墙
systemctl stop firewalld.service


# 禁用 firewall 开机启动
systemctl disable firewalld.service
systemctl list-unit-files |grep firewalld
```

12.2.4 调整系统参数

请按照下面的方法进行操作系统内核参数调整，参数数值一般来说是一个固定的取值，由 Oracle 官方给出建议。请读者严格按照如下数值进行设置。

```
vi /etc/sysctl.conf ,add line similar to the following.
fs.file-max = 6815744
kernel.sem = 250 32000 100 128
kernel.shmmni = 4096
kernel.shmall = 1073741824
kernel.shmmax = 85899345920
kernel.panic_on_oops = 1
net.core.rmem_default = 262144
net.core.rmem_max = 4194304
net.core.wmem_default = 262144
net.core.wmem_max = 1048576
net.ipv4.conf.all.rp_filter = 2
```

```
net.ipv4.conf.default.rp_filter = 2
fs.aio-max-nr = 1048576
net.ipv4.ip_local_port_range = 9000 65500
vm.nr_hugepages = 32768  #按实际需要修改，值=SGA(MB)/2+10

#使修改生效
/sbin/sysctl -p

#确认修改成功
/sbin/sysctl -a
```

调整 Shell 资源限制：

```
vi /etc/security/limits.conf       #添加如下行：
grid soft nproc 4047
grid hard nproc 16384
grid soft nofile 4096
grid hard nofile 65536
oracle soft nproc 131072
oracle hard nproc 131072
oracle soft nofile 40964
oracle hard nofile 65536
oracle soft stack  10240
oracle hard stack  32768
oracle soft memlock 26843545
oracle hard memlock 26843545
```

配置网络 NOZEROCONF 参数：

```
vi /etc/sysconfig/network    #添加如下一行：
NOZEROCONF=yes
```

修改/etc/pam.d/login：

```
vi /etc/pam.d/login #添加或编辑下面一行内容：
session required pam_limits.so
```

关闭 SELinux：

```
#查看 SELinux 状态：
1、/usr/sbin/sestatus -v        ##如果 SELinux status 参数为 enabled 即为开启状态
SELinux status:                 enabled
2、getenforce                  ##也可以用这个命令检查

关闭 SELinux：
1、临时关闭（不用重启机器）：
setenforce 0                   ##设置 SELinux 成为 permissive 模式
                            ##setenforce 1 设置 SELinux 成为 enforcing 模式
2、修改配置文件需要重启机器：

修改/etc/selinux/config 文件
将 SELINUX=enforcing 改为 SELINUX=disabled
重启机器即可
```

12.2.5 修改 hostname

在启动图形界面安装的时候，安装程序会首先检查 hostname 设置是否正确。

```
vi /etc/sysconfig/network
HOSTNAME=san-rac14
```

12.2.6 配置 hosts

/etc/hosts 示例：

```
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
10.0.0.17 san-rac14
10.0.0.18 san-rac15
10.0.0.19 san-rac14-vip
10.0.0.20 san-rac15-vip
192.168.1.10 san-rac14-priv
192.168.1.11san-rac15-priv
10.0.0.21  abcsdb-scan
```

12.2.7 创建用户和组

创建用户是软件安装的必要步骤，强烈建议按照 Oracle 官方的标准进行配置，数据库软件用户为 oracle、集群软件为 grid。当然，如果读者一定要使用其他用户来安装也并非不可，但请注意用户的属组设置正确。

使用 root 用户登录，创建组：

```
groupadd -g 54321 oinstall
groupadd -g 54322 dba
groupadd -g 54323 oper
groupadd -g 54324 backupdba
groupadd -g 54325 dgdba
groupadd -g 54326 kmdba
groupadd -g 54327 asmdba
groupadd -g 54328 asmoper
groupadd -g 54329 asmadmin
groupadd -g 54330 racdba
创建 grid 和 oracle 用户
/usr/sbin/useradd -u 54321 -g oinstall -G
dba,asmdba,backupdba,dgdba,kmdba,racdba,oper oracle
/usr/sbin/useradd -u 54322 -g oinstall -G dba,asmadmin,asmdba,asmoper grid
```

创建完后使用 root 用户通过 passwd 命令修改用户口令：

```
passwd oracle
passwd grid
```

12.2.8 挂载安装目录

下面将开始创建安装目录，也是就是软件安装的位置，存储空间。

```
df -lh
fdisk -l | grep sd
NEED REPLACE

pvcreate  /dev/sdb

#创建 VG
vgcreate -s 32m /dev/sdb oraclevg
vgcreate -s 32m oraclevg /dev/sdb

#创建 LV
lvcreate -L 90G -n lv_oracle oraclevg
lvcreate -L 89G -n lv_oracle oraclevg

#创建 FS
mkfs.xfs /dev/mapper/oraclevg-lv_oracle

#挂载、配置开机自动挂载
vi /etc/fstab
NEED REPLACE
mkdir /u01
mount /u01
```

12.2.9 创建安装目录

开始创建安装目录：

```
mkdir -p /oracle/app/12.2.0/grid
mkdir -p /oracle/app/grid
mkdir -p /oracle/app/oracle
chown -R grid:oinstall /oracle
chown oracle:oinstall /oracle/app/oracle
chmod -R 775 /oracle/
```

12.2.10 设置用户环境变量

oracle 和 grid 用户需要提前设置好环境变量，以便安装。该步骤并非此时此刻必须设置完毕，但强烈建议在安装之前准备妥当，以便安装，并且也有利于后期的 Oracle 软件使用。

```
su - grid
echo "                     "                        >>.bash_profile
echo "export ORACLE_BASE=/u01/app/grid"                  >>.bash_profile
echo "export GI_HOME=/u01/app/12.2.0.1/grid"            >>.bash_profile
echo "export ORACLE_SID=+ASM1"                        >>.bash_profile
echo "export ORACLE HOME=/u01/app/12.2.0.1/grid"        >>.bash profile
echo "export NLS LANG=American America.ZHS16GBK"        >>.bash profile
echo "export TMP=/tmp"                               >>.bash_profile
```

```
echo "export TMPDIR=/tmp"                                  >>.bash_profile
echo "export ORACLE_TERM=vt100"                            >>.bash_profile
echo "export OPATCH PLATFORM ID=226"                       >>.bash profile
echo "export PATH=\$GI HOME/bin:\$GI HOME/OPatch:\$PATH"  >>.bash profile
echo "set -o vi"                                           >>.bash_profile
echo "export EDITOR=vi"                                    >>.bash_profile
echo "NAME=\`hostname\`"                                   >>.bash_profile
echo "umask 022"                                           >>.bash profile
echo "stty erase ^H"                                       >>.bash profile
su - oracle
echo "                    "                                >>.bash_profile
echo "export ORACLE_BASE=/u01/app/oracle"                  >>.bash_profile
echo "export ORACLE_SID=iacore1"                           >>.bash_profile
echo "export ORACLE HOME=/u01/app/oracle/product/12.2.0.1/dbhome 1"
>>.bash profile
echo "export GI_HOME=/u01/app/12.2.0.1/grid"               >>.bash_profile
echo "export NLS_LANG=American_America.ZHS16GBK"           >>.bash_profile
echo "export TMP=/tmp"                                     >>.bash_profile
echo "export TMPDIR=/tmp"                                  >>.bash_profile
echo "export ORACLE TERM=vt100"                            >>.bash profile
echo "export OPATCH PLATFORM ID=226"                       >>.bash profile
echo "export PATH=\$ORACLE_HOME/bin:\$ORACLE_HOME/OPatch:\$PATH" >>.bash_profile
echo "set -o vi"                                           >>.bash_profile
echo "export EDITOR=vi"                                    >>.bash_profile
echo "NAME=\`hostname\`"                                   >>.bash_profile
echo "umask 022"                                           >>.bash profile
echo "stty erase ^H"                                       >>.bash_profile
```

12.2.11 配置共享存储

在 Red Hat Enterprise Linux (RHEL) 6 以前，Oracle 都是采用 ASMLib 包来管理 ASM 磁盘。但是 2011 年 5 月甲骨文发表了一份声明，声明中称 Oracle 将不再提供 Red Hat Enterprise Linux (RHEL) 6 的 ASMLib 包和相关更新。

甲骨文在这份声明中表示，ASMLib 更新将通过 Unbreakable Linux Network（ULN）来发布，并仅对 Oracle Linux 客户开放。ULN 虽然为甲骨文和红帽的客户服务，但如果客户想要使用 ASMLib，就必须使用 Oracle 的 kernel 来替换掉红帽的 kernel。

假设/dev/sd[e,f,d,g,h,j,k,i,l] 为共享磁盘 UDEV 方式配置共享存储。首先计算磁盘的 scsi_id：

```
[root@zj-vsan-rac14 ~]# /usr/lib/udev/scsi id --whitelisted --replace-whitespace
--device=/dev/sde
36000c29d62e841646c62666fdf2d452f
[root@zj-vsan-rac14 ~]# /usr/lib/udev/scsi_id --whitelisted --replace-whitespace
--device=/dev/sdf
36000c29848594122e8288dfbfc40c920
[root@zj-vsan-rac14 ~]# /usr/lib/udev/scsi id --whitelisted --replace-whitespace
--device=/dev/sdd
36000c29c04e6369cded5ed2340a5d558
```

```
[root@zj-vsan-rac14 ~]# /usr/lib/udev/scsi_id --whitelisted --replace-whitespace
--device=/dev/sdg
36000c293f2a1f103e15de4c4a2eff778
[root@zj-vsan-rac14 ~]# /usr/lib/udev/scsi_id --whitelisted --replace-whitespace
--device=/dev/sdh
36000c29a185c13594dcf8e148473b180
[root@zj-vsan-rac14 ~]# #ocr_vote
[root@zj-vsan-rac14 ~]# /usr/lib/udev/scsi_id --whitelisted --replace-whitespace
--device=/dev/sdj
36000c29c0bc34f24666ce42853c34584
[root@zj-vsan-rac14 ~]# /usr/lib/udev/scsi_id --whitelisted --replace-whitespace
--device=/dev/sdk
36000c29c0529c46a96a9d82715b92c4d
[root@zj-vsan-rac14 ~]# /usr/lib/udev/scsi_id --whitelisted --replace-whitespace
--device=/dev/sdi
36000c29991476544d0916b7e041e1c55
[root@zj-vsan-rac14 ~]# #redo
[root@zj-vsan-rac14 ~]# /usr/lib/udev/scsi_id --whitelisted --replace-whitespace
--device=/dev/sdl
36000c292a10d3f682d7ba110f3554a38
```

下面进行磁盘权限绑定：

```
vi /etc/udev/rules.d/99-my-asmdevices.rules  #添加如下行
KERNEL=="sd*[!0-9]", ENV{DEVTYPE}=="disk", SUBSYSTEM=="block",
PROGRAM=="/usr/lib/udev/scsi_id -g -u -d $devnode",
RESULT=="36000c29991476544d0916b7e041e1c55", RUN+="/bin/sh -c 'mknod
/dev/OCR_VOTE_1 b $major $minor; chown grid:asmadmin /dev/OCR_VOTE_1; chmod 0660
/dev/OCR_VOTE_1'"
KERNEL=="sd*[!0-9]", ENV{DEVTYPE}=="disk", SUBSYSTEM=="block",
PROGRAM=="/usr/lib/udev/scsi_id -g -u -d $devnode",
RESULT=="36000c29c0529c46a96a9d82715b92c4d", RUN+="/bin/sh -c 'mknod
/dev/OCR_VOTE_2 b $major $minor; chown grid:asmadmin /dev/OCR_VOTE_2; chmod 0660
/dev/OCR_VOTE_2'"
KERNEL=="sd*[!0-9]", ENV{DEVTYPE}=="disk", SUBSYSTEM=="block",
PROGRAM=="/usr/lib/udev/scsi_id -g -u -d $devnode",
RESULT=="36000c29c0bc34f24666ce42853c34584", RUN+="/bin/sh -c 'mknod
/dev/OCR_VOTE_3 b $major $minor; chown grid:asmadmin /dev/OCR_VOTE_3; chmod 0660
/dev/OCR_VOTE_3'"

KERNEL=="sd*[!0-9]", ENV{DEVTYPE}=="disk", SUBSYSTEM=="block",
PROGRAM=="/usr/lib/udev/scsi_id -g -u -d $devnode",
RESULT=="36000c293f2a1f103e15de4c4a2eff778", RUN+="/bin/sh -c 'mknod /dev/DATA_01
b $major $minor; chown grid:asmadmin /dev/DATA_01; chmod 0660 /dev/DATA_01'"
KERNEL=="sd*[!0-9]", ENV{DEVTYPE}=="disk", SUBSYSTEM=="block",
PROGRAM=="/usr/lib/udev/scsi_id -g -u -d $devnode",
RESULT=="36000c29848594122e8288dfbfc40c920", RUN+="/bin/sh -c 'mknod /dev/DATA_02
b $major $minor; chown grid:asmadmin /dev/DATA_02; chmod 0660 /dev/DATA_02'"
KERNEL=="sd*[!0-9]", ENV{DEVTYPE}=="disk", SUBSYSTEM=="block",
PROGRAM=="/usr/lib/udev/scsi_id -g -u -d $devnode",
RESULT=="36000c29a185c13594dcf8e148473b180", RUN+="/bin/sh -c 'mknod /dev/DATA_03
```

```
b $major $minor; chown grid:asmadmin /dev/DATA_03; chmod 0660 /dev/DATA_03'"
KERNEL=="sd*[!0-9]", ENV{DEVTYPE}=="disk", SUBSYSTEM=="block",
PROGRAM=="/usr/lib/udev/scsi id -g -u -d $devnode",
RESULT=="36000c29c04e6369cded5ed2340a5d558", RUN+="/bin/sh -c 'mknod /dev/DATA 04
b $major $minor; chown grid:asmadmin /dev/DATA_04; chmod 0660 /dev/DATA_04'"
KERNEL=="sd*[!0-9]", ENV{DEVTYPE}=="disk", SUBSYSTEM=="block",
PROGRAM=="/usr/lib/udev/scsi_id -g -u -d $devnode",
RESULT=="36000c29d62e841646c62666fdf2d452f", RUN+="/bin/sh -c 'mknod /dev/ARCH 01
b $major $minor; chown grid:asmadmin /dev/ARCH 01; chmod 0660 /dev/ARCH 01'"

KERNEL=="sd*[!0-9]", ENV{DEVTYPE}=="disk", SUBSYSTEM=="block",
PROGRAM=="/usr/lib/udev/scsi_id -g -u -d $devnode",
RESULT=="36000c292a10d3f682d7ba110f3554a38", RUN+="/bin/sh -c 'mknod /dev/REDO_01
b $major $minor; chown grid:asmadmin /dev/REDO 01; chmod 0660 /dev/REDO 01'"
```

重启 udev，使刚才的操作生效。

```
udevadm trigger --type=devices --action=change
udevadm control --reload
```



Linux 内核 2.6 以上才支持 udev。

12.2.12　禁用 Transparent HugePages

关于 Transparent HugePages 为什么需要关闭的解释，Oracle 官方给出的答案如下：

> Disable Transparent HugePages
>
> Oracle recommends that you disable Transparent HugePages, because they may causes delays in accessing memory that can result in node restarts in Oracle RAC environments, or performance issues or delays for Oracle Database single instances. Oracle continues to recommend using standard HugePages for Linux.

（1）检查是否所有节点已经禁用 Transparent HugePages，在所有节点执行：

```
grep HugePages /proc/meminfo
```

（2）如果 AnonHugePages 不为 0，继续执行下面的操作，修改/etc/grub.conf，添加 transparent_hugepage=never：

```
title Oracle Linux Server (2.6.32-300.25.1.el6uek.x86 64)
root (hd0,0)
kernel /vmlinuz-2.6.32-300.25.1.el6uek.x86_64 ro root=LABEL=/
transparent_hugepage=never
initrd /initramfs-2.6.32-300.25.1.el6uek.x86_64.img
```

（3）修改完成后重启机器 reboot。

12.2.13　配置 NTP 服务

随着企业计算机应用的广度和深度不断加大，网络中的设备种类和业务类型越来越多，服务器的数量也与日俱增。传统上，各种服务器、网络设备使用的时间都是由设备内部时钟来提供的。服务器、网络设备本身的时钟误差是不可避免的，尽管这种误差每天不大，但经过一段时间的累积就会出现大的时间差，从而导致网络中各服务器、网络设备的时间不一致。

对于独立单点运行的服务器或系统来说，这种时间的不一致性不会带来什么问题，然而，对于像 Oracle RAC 集群一类的集群架构而言却是致命的。所以在安装 Oracle RAC 之前，强烈建议服务器配置 NTP 服务，以确保时间的一致性。

```
# cat /etc/ntp.conf
# For more information about this file, see the man pages
# ntp.conf(5), ntp_acc(5), ntp_auth(5), ntp_clock(5), ntp_misc(5), ntp_mon(5).

driftfile /var/lib/ntp/drift

# Permit time synchronization with our time source, but do not
# permit the source to query or modify the service on this system.
restrict default kod nomodify notrap nopeer noquery
restrict -6 default kod nomodify notrap nopeer noquery

# Permit all access over the loopback interface.  This could
# be tightened as well, but to do so would effect some of
# the administrative functions.
restrict 127.0.0.1
restrict -6 ::1

# Hosts on local network are less restricted.
#restrict 192.168.1.0 mask 255.255.255.0 nomodify notrap

# Use public servers from the pool.ntp.org project.
# Please consider joining the pool (http://www.pool.ntp.org/join.html).
#server 0.rhel.pool.ntp.org iburst
#server 1.rhel.pool.ntp.org iburst
#server 2.rhel.pool.ntp.org iburst
#server 3.rhel.pool.ntp.org iburst

#broadcast 192.168.1.255 autokey          # broadcast server
#broadcastclient                          # broadcast client
#broadcast 224.0.1.1 autokey              # multicast server
#multicastclient 224.0.1.1                # multicast client
#manycastserver 239.255.254.254           # manycast server
#manycastclient 239.255.254.254 autokey   # manycast client

# Enable public key cryptography.
#crypto

includefile /etc/ntp/crypto/pw
```

```
# Key file containing the keys and key identifiers used when operating
# with symmetric key cryptography.
keys /etc/ntp/keys

# Specify the key identifiers which are trusted.
#trustedkey 4 8 42

# Specify the key identifier to use with the ntpdc utility.
#requestkey 8

# Specify the key identifier to use with the ntpq utility.
#controlkey 8

# Enable writing of statistics records.
#statistics clockstats cryptostats loopstats peerstats
server 3.9.0.1 version 3
[root@ptscsdb1oracle]#  /sbin/service ntpd  status
ntpd (pid  12809) is running...


--添加-x 选项
[root@ptscsdb2pp ~]# vi /etc/sysconfig/ntpd

# Drop root to id 'ntp:ntp' by default.
OPTIONS="-x -u ntp:ntp -p /var/run/ntpd.pid -g"

[root@ptscsdb2 ~]# /sbin/service ntpd stop
Shutting down ntpd: [  OK  ]
[root@ptscsdb2 ~]# /sbin/service ntpd start
Starting ntpd: [  OK  ]

[root@ptscsdb1~]# ntpq -p
    remote           refid      st t when poll reach   delay   offset  jitter
==========================================================================
*3.9.0.1        .GPS.            1 u   58   64  377    0.412   -2.067   0.608
```

12.2.14 其他节点重复步骤

在其他节点，重复执行上一小节的所有步骤。

12.2.15 互信配置

因为 Oracle 集群软件和数据库软件安装的时候程序会在所有节点之间进行文件的复制传输，所以需要配置节点之间的用户互信，互信的访问特点是不需要密码。以下的互信配置要在所有节点上重复执行，并且在配置完毕之后进行测试确认，否则软件安装将无法进行。

```
su - grid
/u01/app/12.2.0.1/grid/deinstall/sshUserSetup.sh -user grid  -hosts
"zj-vsan-rac14 zj-vsan-rac14-priv zj-vsan-rac15 zj-vsan-rac15-priv" -advanced
```

```
-noPromptPassphrase

##为 oracle 用户配置 ssh
su - oracle
/u01/app/12.2.0.1/grid/deinstall/sshUserSetup.sh -user oracle  -hosts
"zj-vsan-rac14 zj-vsan-rac14-priv zj-vsan-rac15 zj-vsan-rac15-priv" -advanced
-noPromptPassphrase

#测试
su - oracle
ssh zj-vsan-rac14 date
ssh zj-vsan-rac15 date
ssh zj-vsan-rac14-priv date
ssh zj-vsan-rac15-priv date
su - grid
ssh zj-vsan-rac14 date
ssh zj-vsan-rac15 date
ssh zj-vsan-rac14-priv date
ssh zj-vsan-rac15-priv date
```

12.3 GRID 安装

解压 GRID 安装包到/u01/app/12.2.0.1/grid 下：

```
su - grid
./gridSetup.sh
```

12.3.1　安装前预先检查

以下脚本 runcluvfy.sh 是 Oracle 官方提供给用户的环境检查脚本，在正式安装之前，通过该脚本可以检查哪些准备工作已经进行完毕，并列出未完成项。

```
./runcluvfy.sh stage -pre crsinst -n ptscsdb1,ptscsdb2 -verbose
```

12.3.2　cvuqdisk 包安装（两个节点都安装）

这是一个在 UNIX 环境下单独的软件包，一般在操作系统的光盘镜像中是没有的，由 Oracle 提供，并且保存在解压之后的/oracle/app/12.2.0/grid/cv/rpm 目录中。

```
[root@ptscsdb1~]# cd /oracle/app/12.2.0/grid/cv/rpm
[root@ptscsdb1rpm]# ll
total 12
-rw-r--r-- 1 grid oinstall 8860 Jan  5 17:36 cvuqdisk-1.0.10-1.rpm
[root@ptscsdb1rpm]#
[root@ptscsdb1rpm]#
[root@ptscsdb1rpm]# rpm -ivh cvuqdisk-1.0.10-1.rpm
Preparing...             ########################################### [100%]
```

```
   1:cvuqdisk              ########################################### [100%]
[root@ptscsdb1rpm]#
```

12.4 开始安装

在以上所有准备工作全部完成之后，开始启动图形化的安装工具，进行正式的安装过程。

```
[rootroot@ptscsdb1]#xhost +
[grid@ptscsdb1grid]$GRID_HOME/gridSetup.sh
```

1. 选择操作类型

如图 12-1 所示，选择第一项，并单击 Next 按钮。

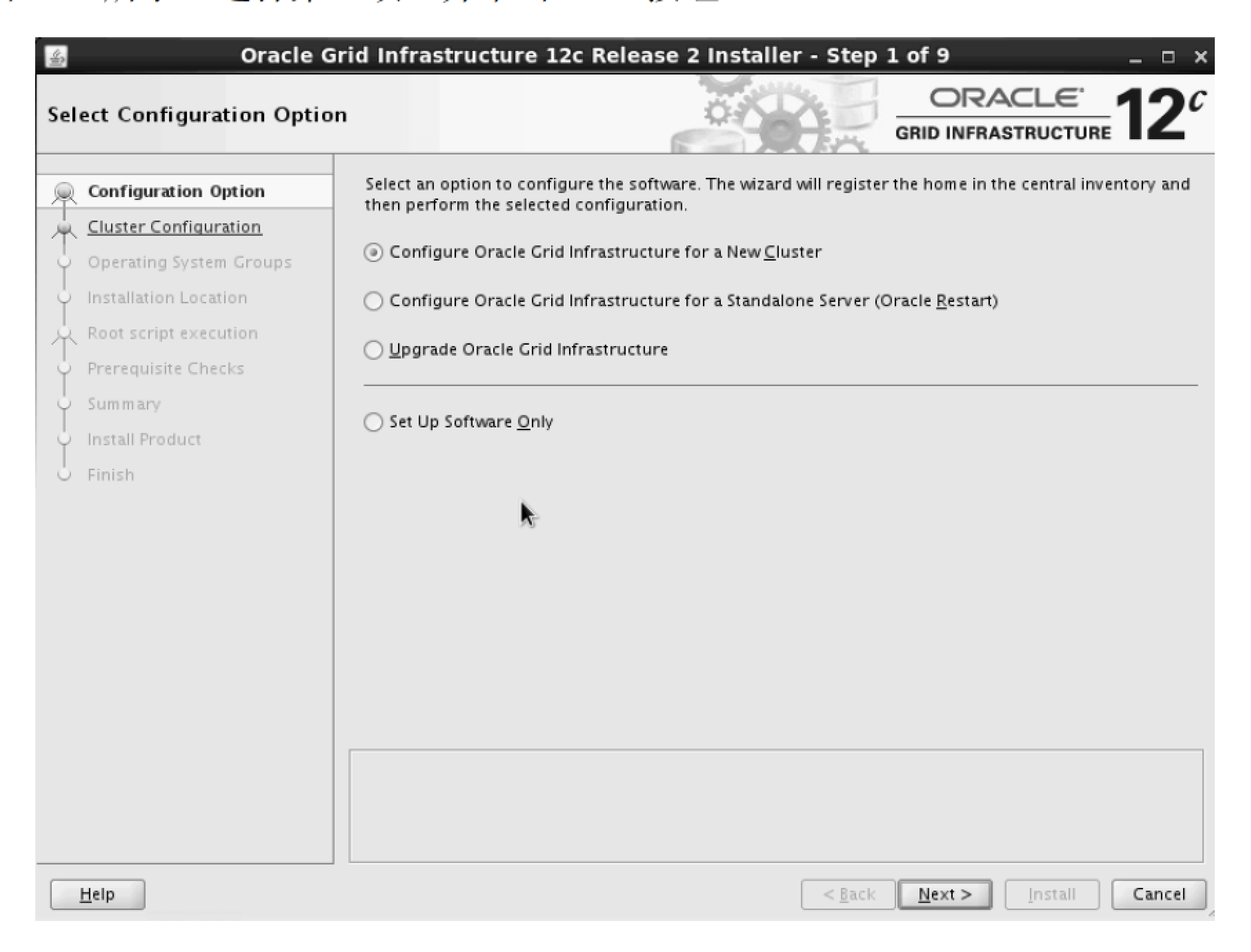


图 12-1　选择操作类型

2. 选择安装集群类型

如图 12-2 所示，同样选择第一项，创建一般的 Oracle 集群，并单击 Next 按钮。

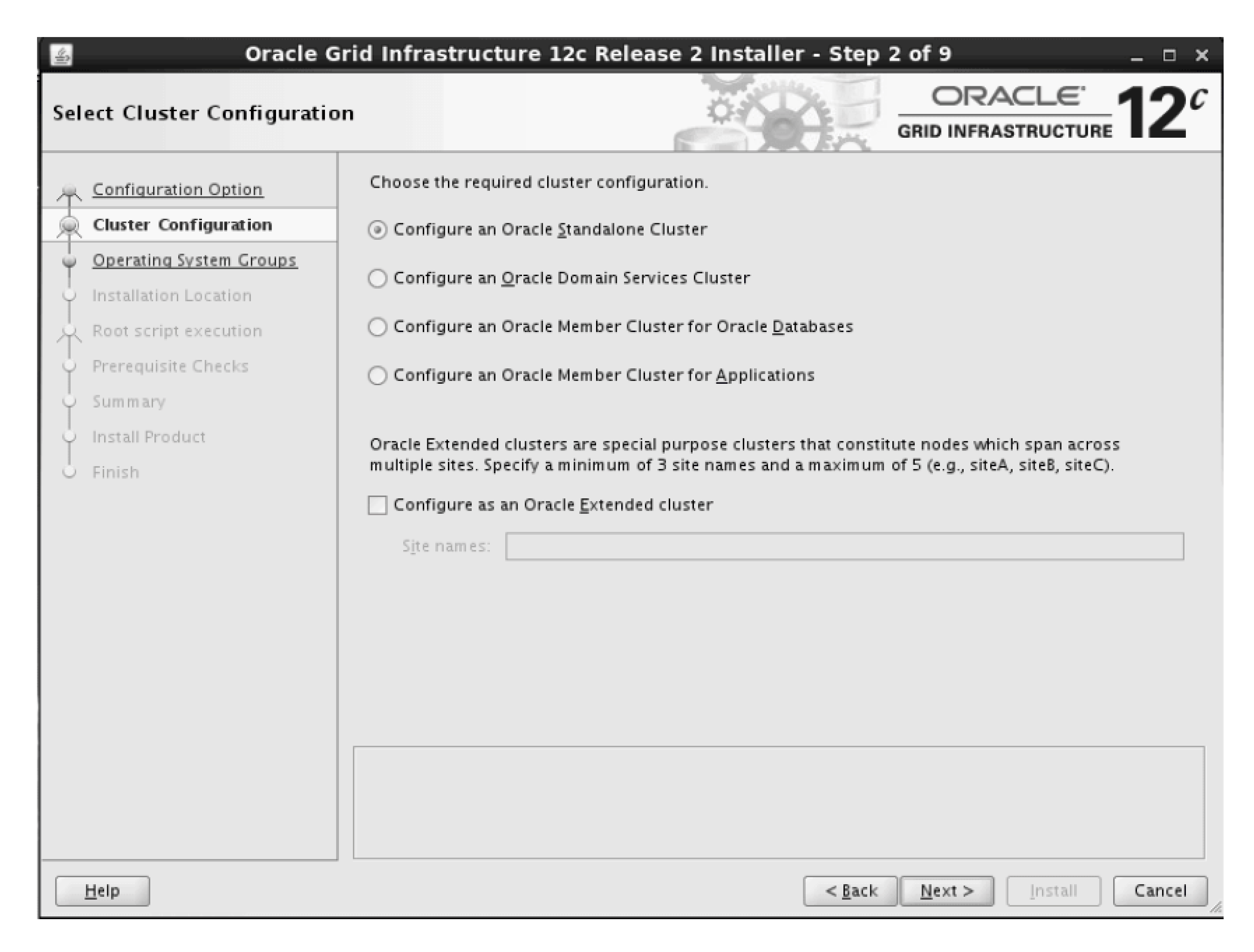


图 12-2　选择安装集群类型

3. 配置集群信息

如图 12-3 所示，为即将安装的集群起个名字，并填写 SCAN Name，这里的 SCAN Name 是配置在 hostname 文件中的，对应一个 IP 地址。SCAN（Single Client Access Name）是 Oracle 从 11g R2 开始推出的，客户端可以通过 SCAN 特性负载均衡地连接到 RAC 数据库。所以在 Oracle 11g R2 中引入了 SCAN（Single ClientAccess Name）的特性。SCAN 是一个域名，可以解析至少 1 个 IP、最多 3 个 SCAN IP，客户端可以通过这个 SCAN 名字来访问数据库。另外，SCAN IP 必须与 public IP 和 VIP 在一个子网。

SCAN 提供一个域名来访问 RAC，域名可以解析 1~3 个（注意，最多 3 个）SCAN IP，我们可以通过 DNS 或者 GNS 来解析实现。其中 DNS 大家都很熟悉，这里不多说。GNS（Grid Naming Service）则是 Oracle 11g R2 的新功能，可以通过 DHCP 服务为节点和 SCAN 分配 VIP 和 SCAN IP。另外，还有一个优点，即对于新加入集群的节点，它会自动分配 VIP 地址，更新集群资源，客户端依然通过 SCAN 特性负载均衡地连接到新增集群节点上。DNS 和 GNS 配置与解析相关内容在下面还有说明。

除了 DNS 和 GNS 解析方法外，SCAN 也可以使用 hosts 文件来解析，但用过的人都知道，此方法不仅在安装 RAC 的时候产生问题，后期使用也是存在问题的，比如 SCAN 域名只能定义一个 SCAN IP。所以这种方法也是 Oracle 不推荐使用的。

尽管如此，很多生产上依然这样使用，也就是废弃了 11g 的新特性 SCAN，而是依然采用 VIP 连接方式。继续单击 Next 按钮。

图 12-3　配置集群信息

4. 添加集群节点信息及配置 SSH 连通性

如图 12-4 所示，请将所有节点的信息填写完整，这里可以进行主机互信的配置，因为上文中已经配置了主机的互信，这里可以单击 Next 按钮，进入下一步。

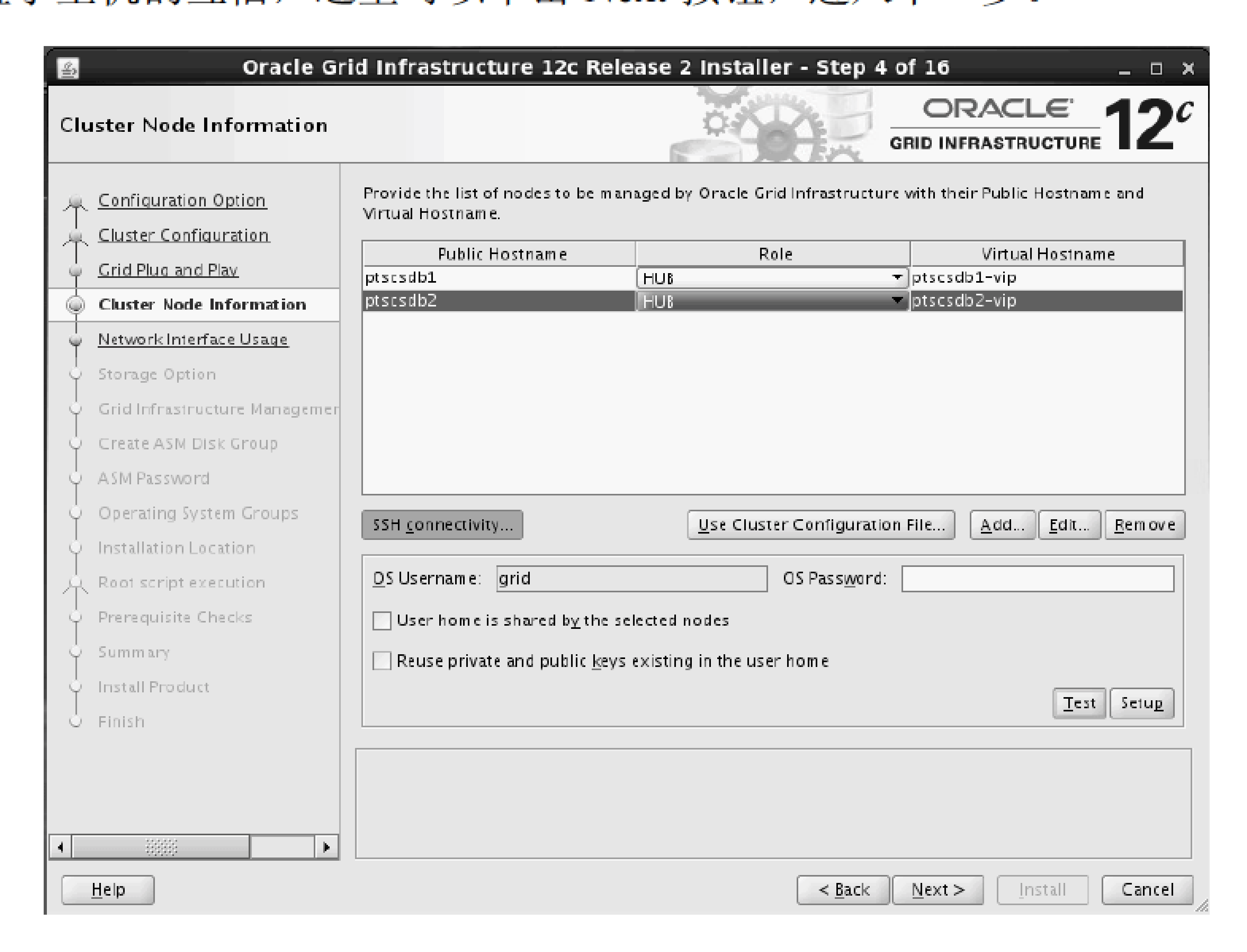


图 12-4　互信和节点的配置

5. 配置网络信息

如图 12-5 所示，指定正确的网卡和 IP 地址，一般来说 192 开头的地址都是集群节点直接互相通信的私有地址，是不对外开放的，其他地址属于公共 IP，单击 Next 按钮进入下一步。

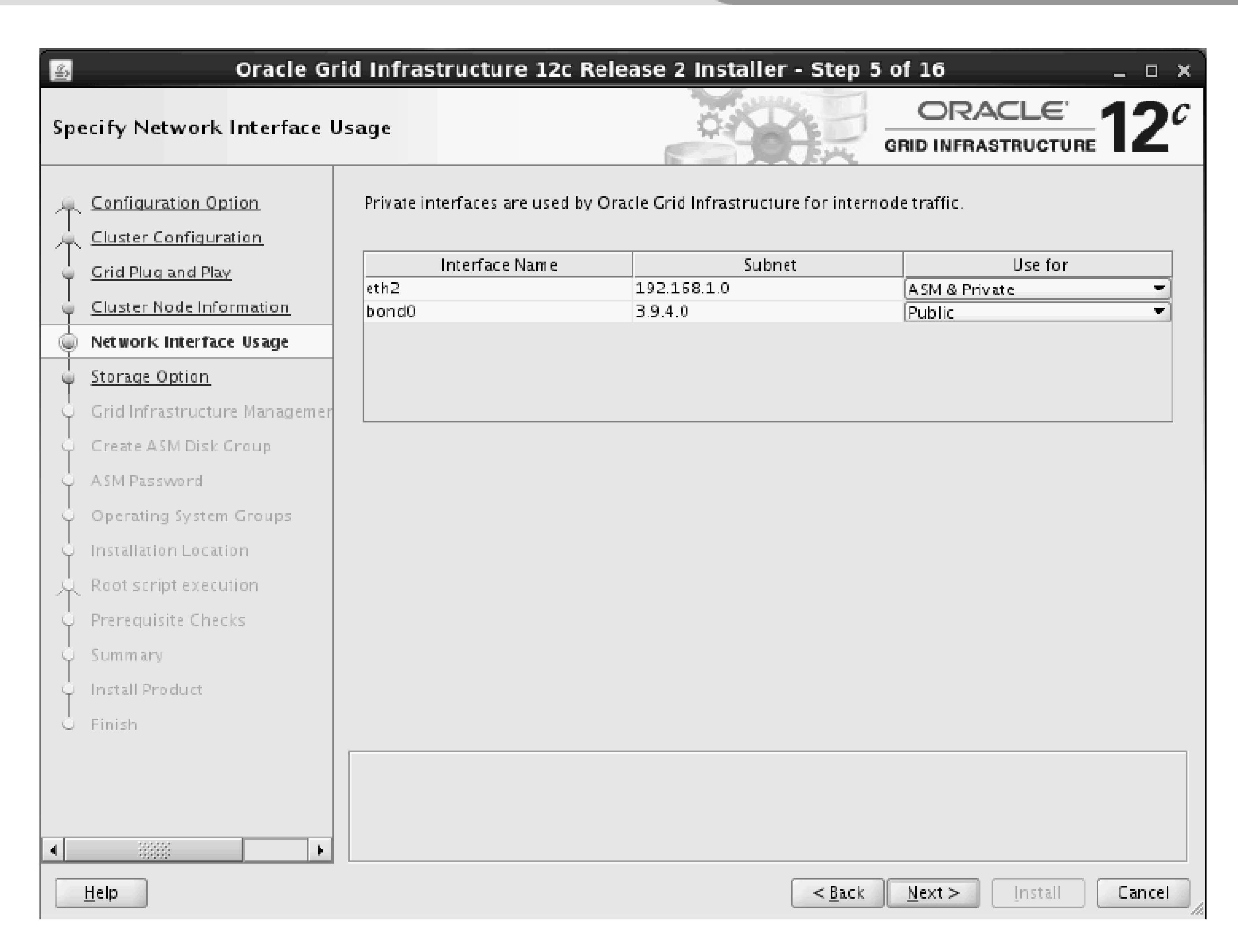


图 12-5　配置网络信息

6. 配置 ASM 方式

如图 12-6 所示，请选择第一项，并单击 Next 按钮。

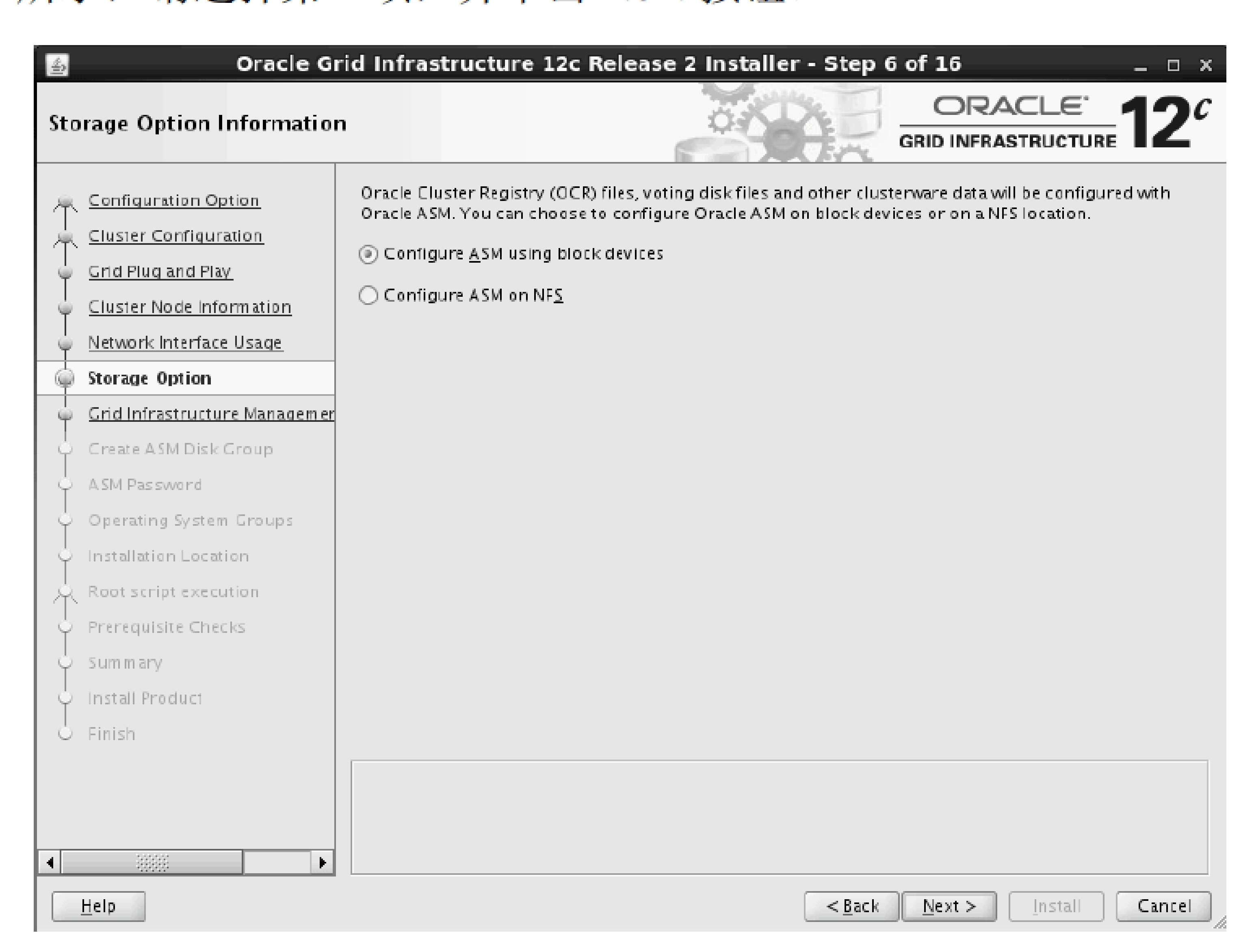


图 12-6　配置 ASM 方式

7. GIMR 磁盘组选项

如图 12-7 所示，请选择“NO”，并单击 Next 按钮。

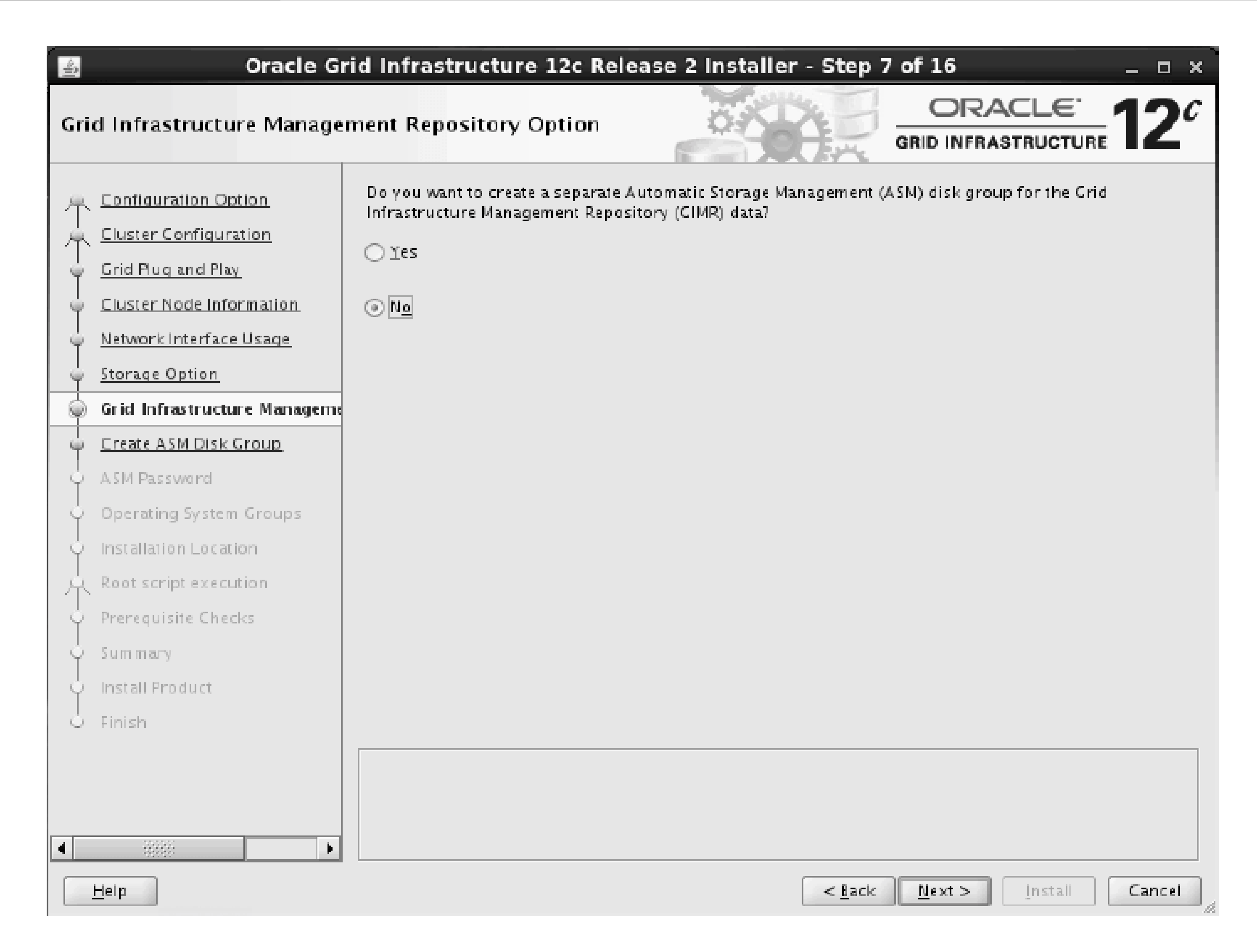


图 12-7　GIMR

8. 配置 OCR 磁盘组

如图 12-8 所示，首先需要给磁盘组起一个名字，这个名字应该提前进行规划，Oracle 并没有一个统一的建议，请根据需要填写，并选择磁盘组包含的磁盘。但是这里要注意，该步骤创建的是存储 OCR 和投票信息的磁盘组，不同于业务数据磁盘组，后者将单独创建。单击 Next 按钮继续下一步。

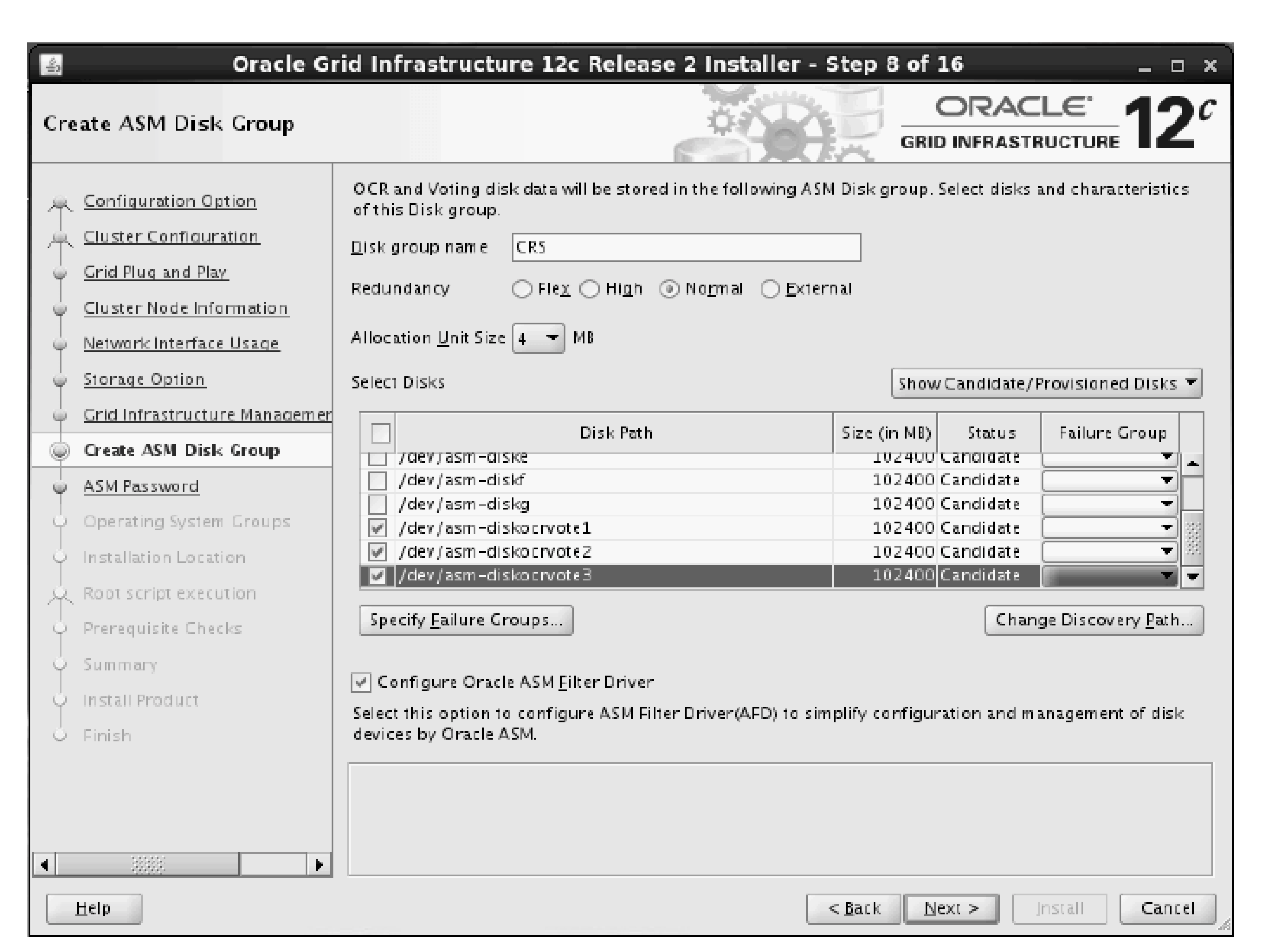


图 12-8　配置 OCR 磁盘组

9. 配置 ASM 密码

如图 12-9 所示，为 ASM 管理员设置密码，并单击 Next 按钮继续下一步。

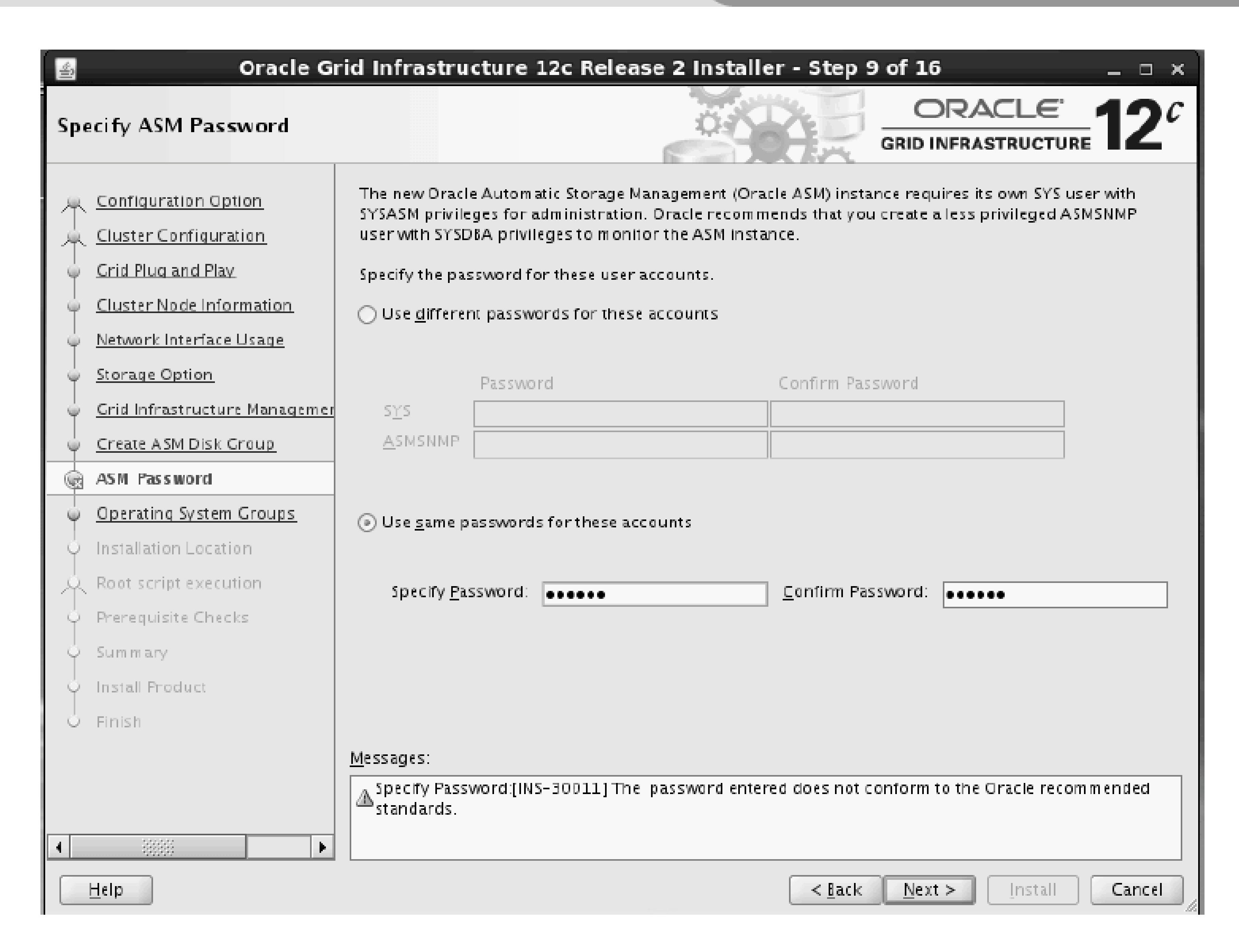


图 12-9　配置 ASM 密码

10. 选择 IPMI

如图 12-10 所示，选择第二项，并单击 Next 按钮继续下一步。

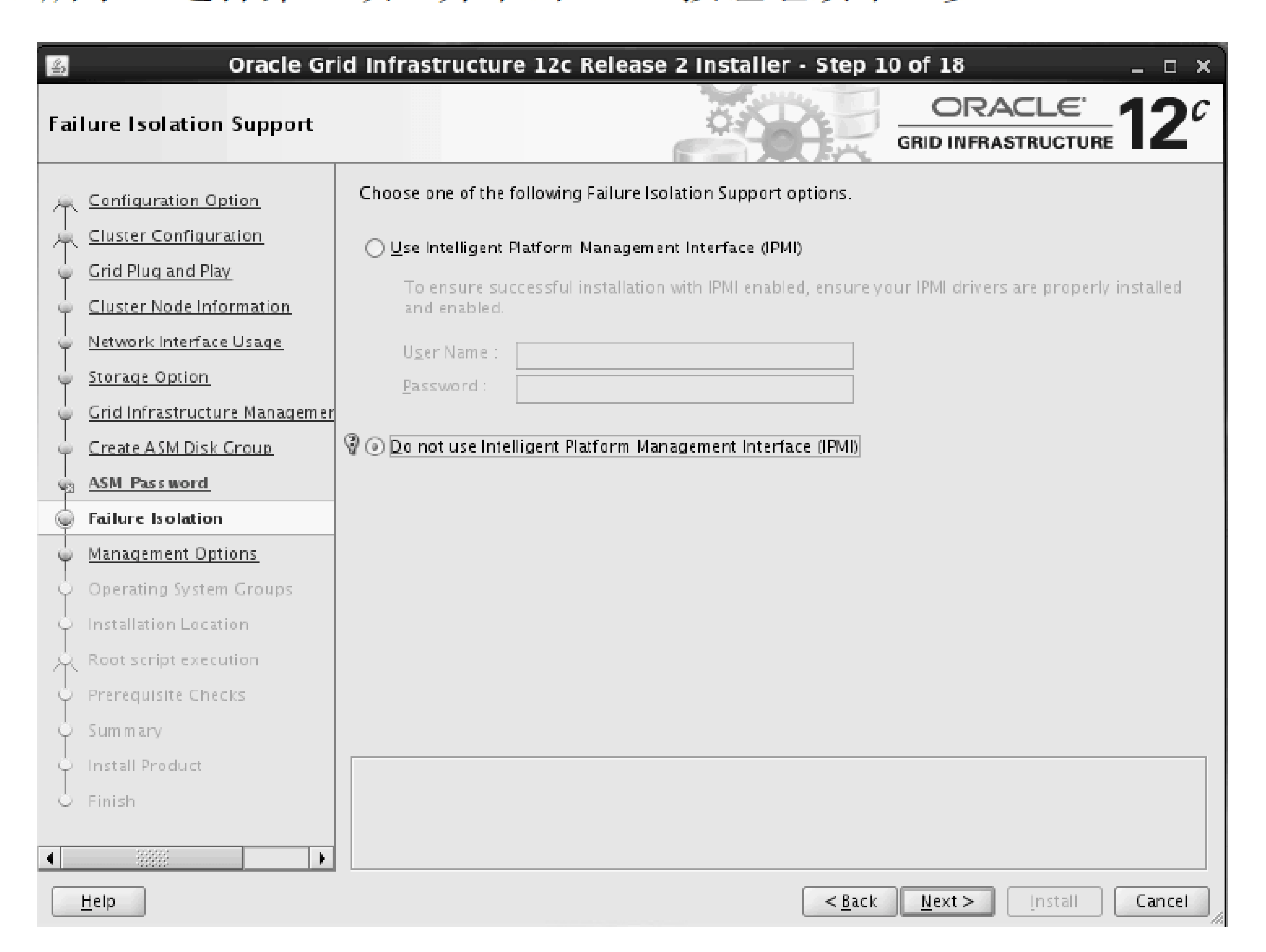


图 1210　IPMI

11. 指定管理选项

如图 12-11 所示，这里请不要选择配置项，并单击 Next 按钮继续下一步。

图 12-11　指定管理选项

12. 配置用户组

如图 12-12 所示，请选择用户的组，并单击 Next 按钮继续下一步。

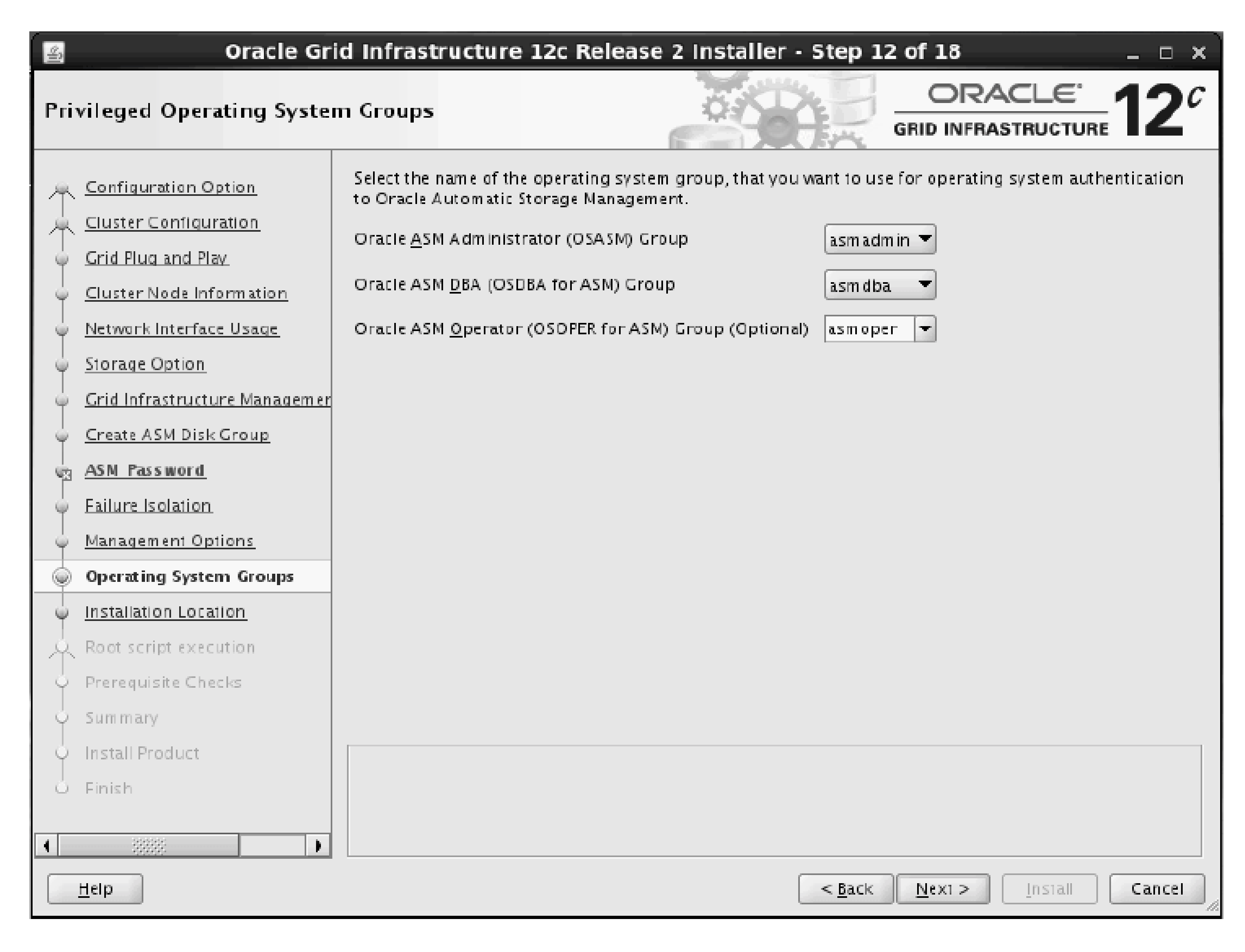


图 12-12　配置用户组

13. 指定安装目录

如图 12-13 所示，请填写集群软件的安装目录，如果 grid 用户的环境变量配置正确，安装程序会自动填写这部分的内容，单击 Next 按钮继续下一步。

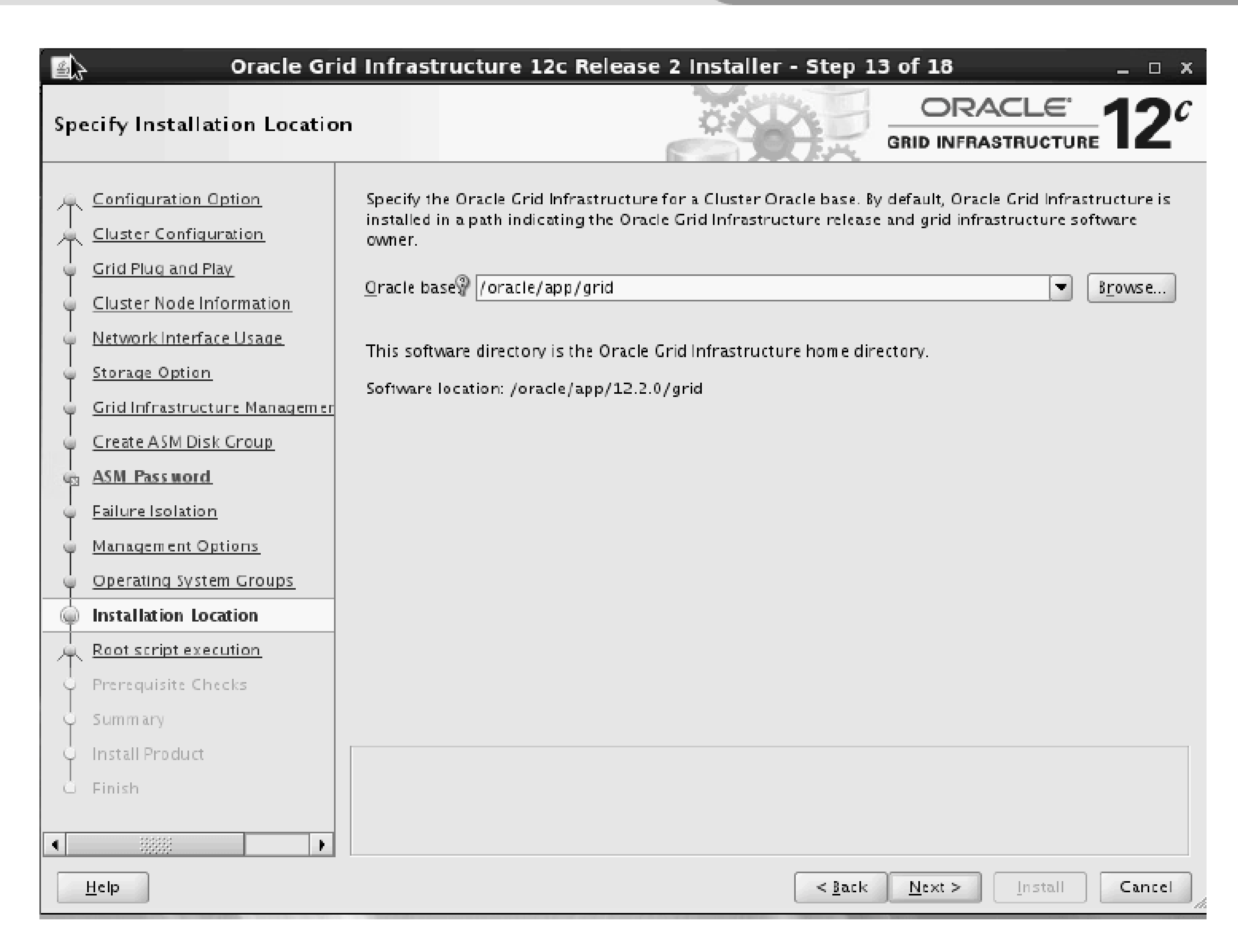


图 12-13　指定安装目录

14. root 脚本执行配置

如图 12-14 所示，请不要选择任何一项，并单击 Next 按钮继续下一步。

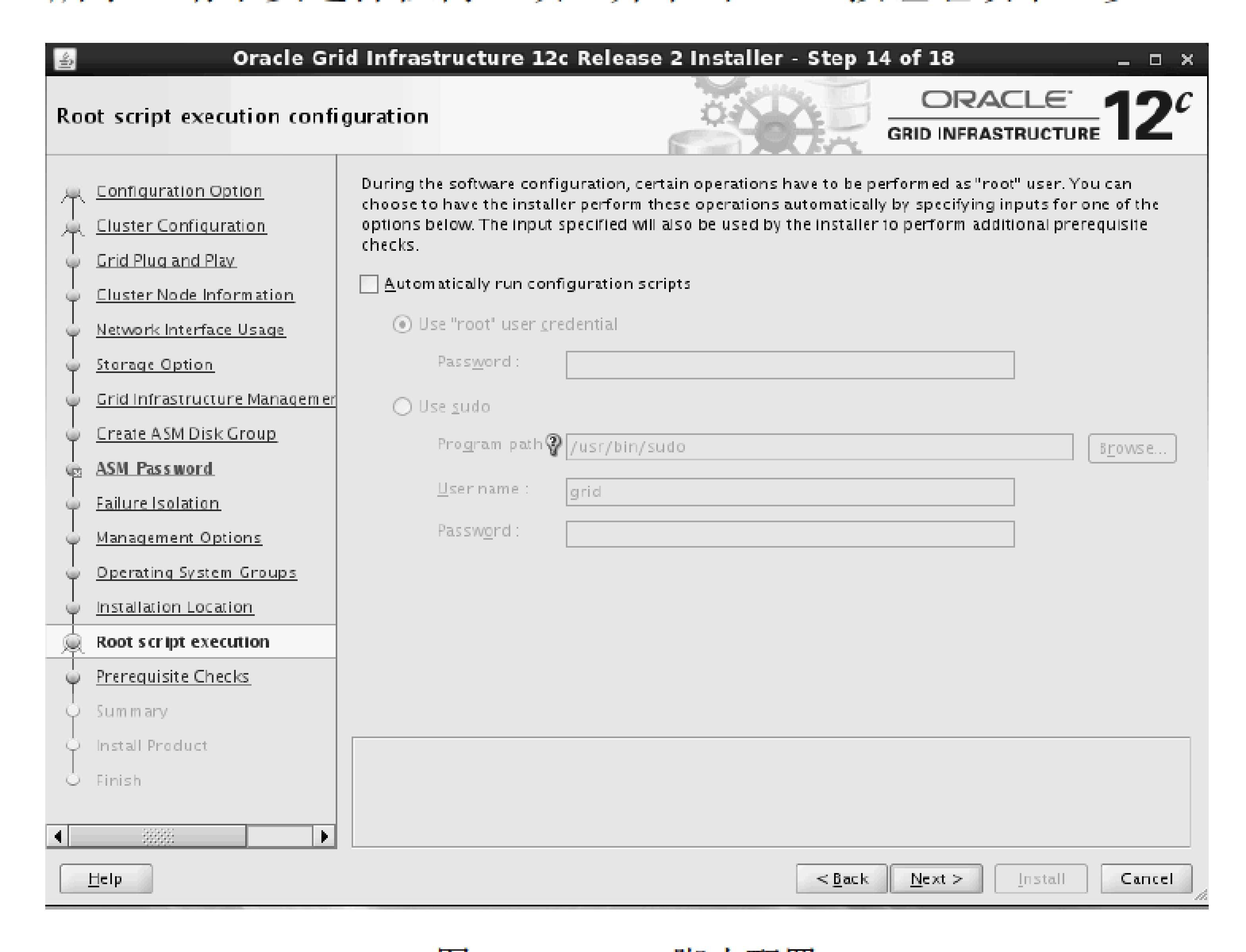


图 12-14　root 脚本配置

15. 安装预检查

如图 12-15 所示，安装程序开始进行最后的环境检查工作。

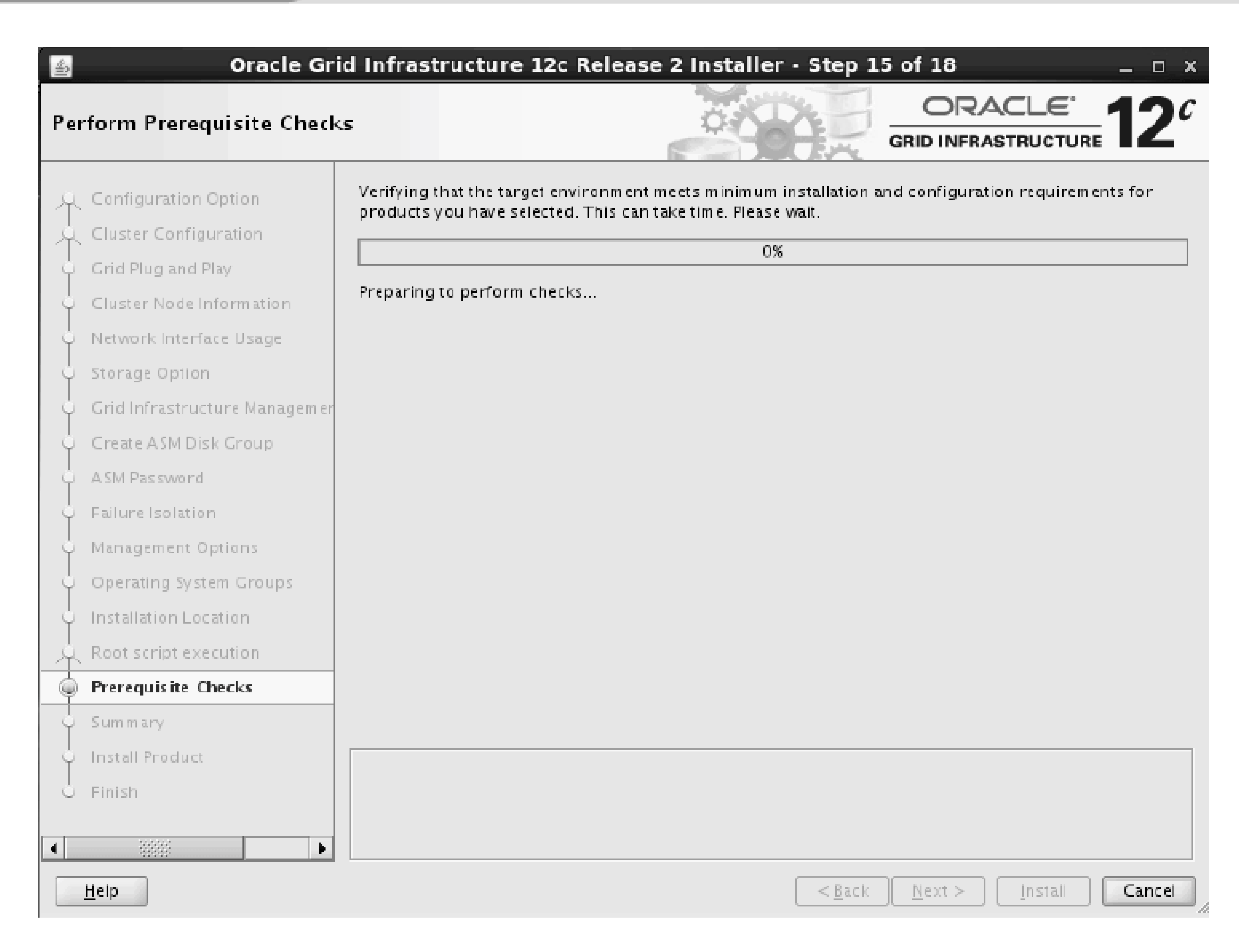


图 12-15　环境检查

如图 12-16 所示，请选择右上角的 Ignore ALL，并单击 Next 按钮继续下一步。

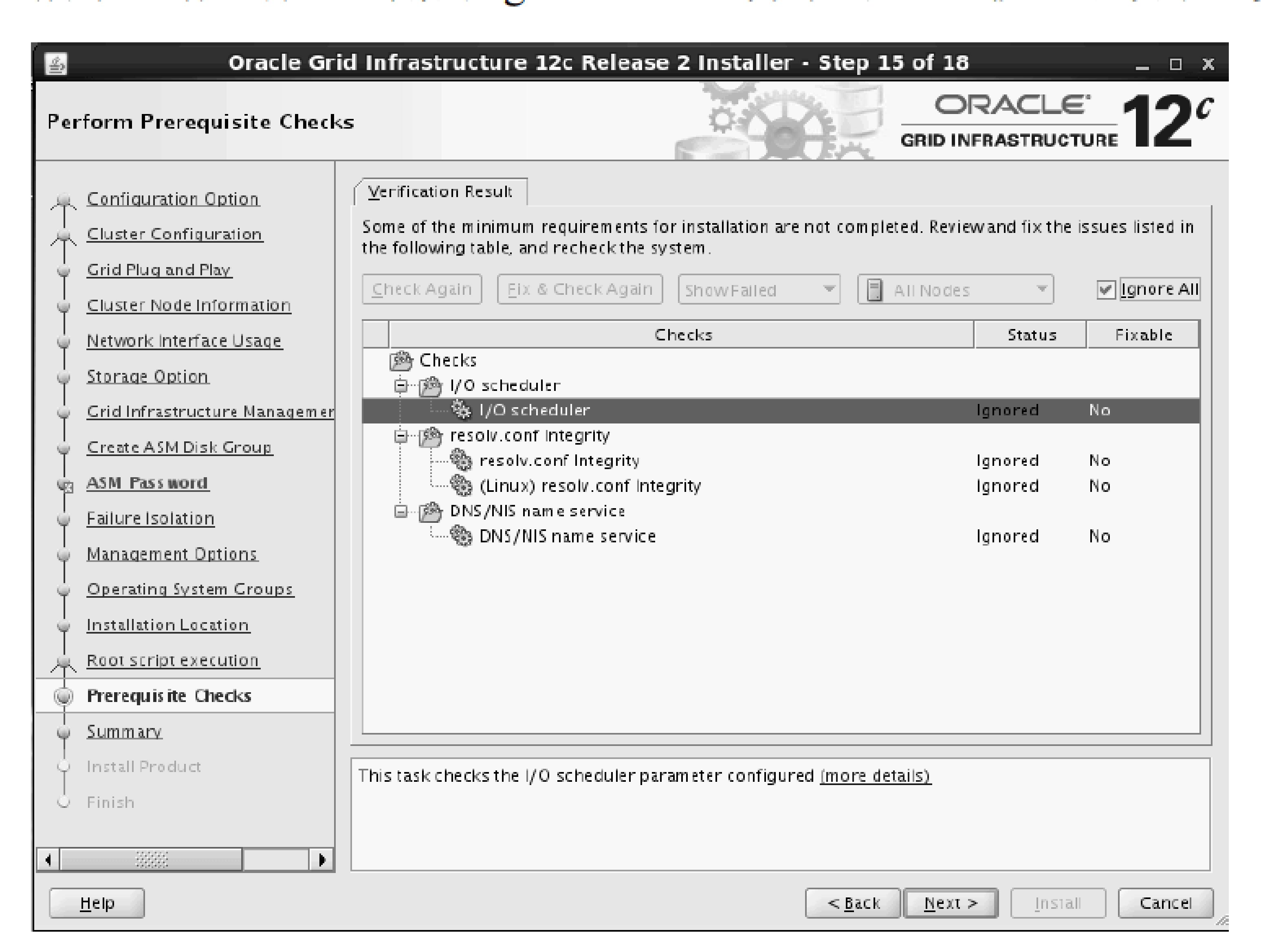


图 12-16　检查结果确认

16. 安装概要

如图 12-17 所示，单击 Install 按钮开始安装。

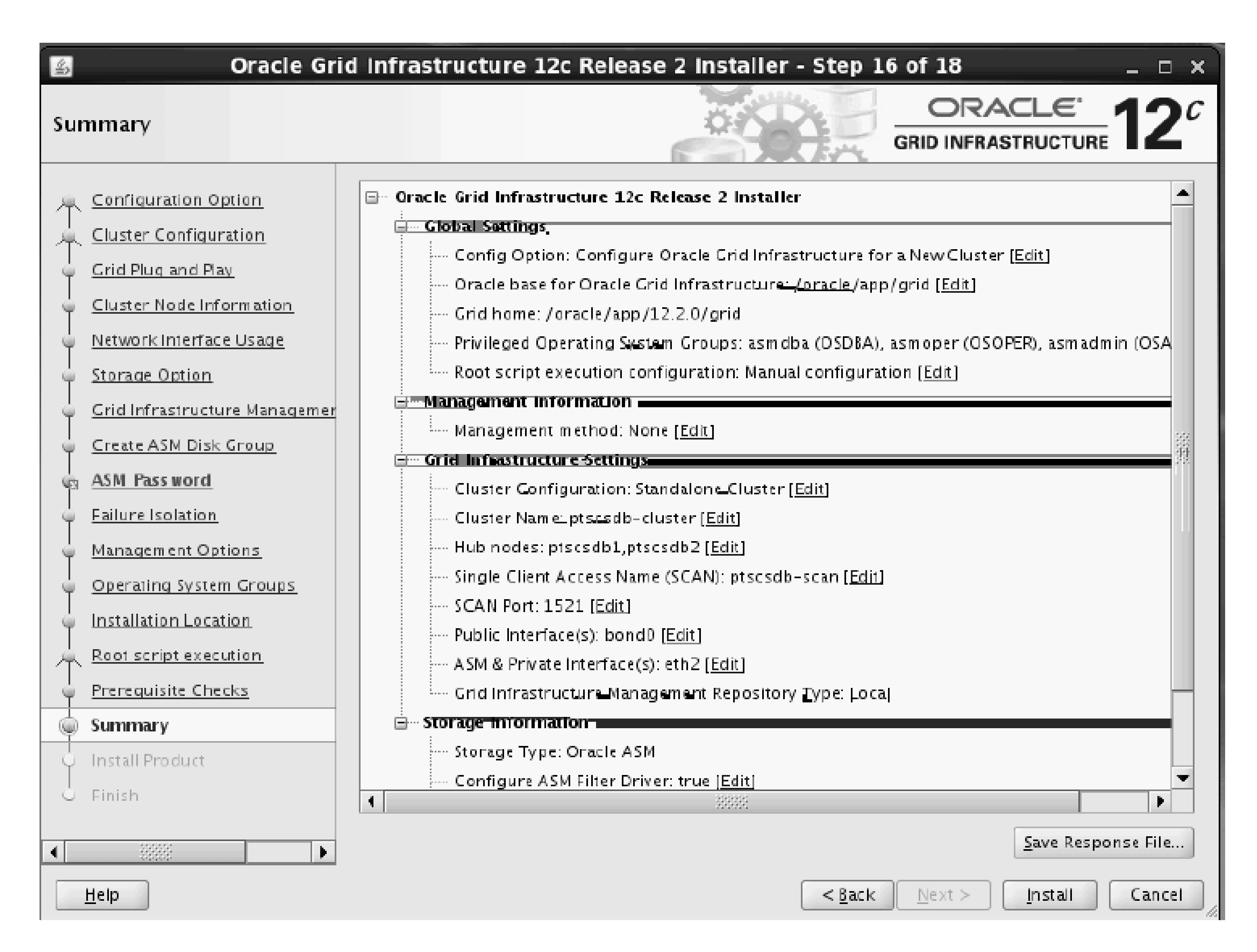


图 12-17　安装概要

如图 12-18 所示，安装程序会显示具体的安装进度和全部的安装项目继续下一步。

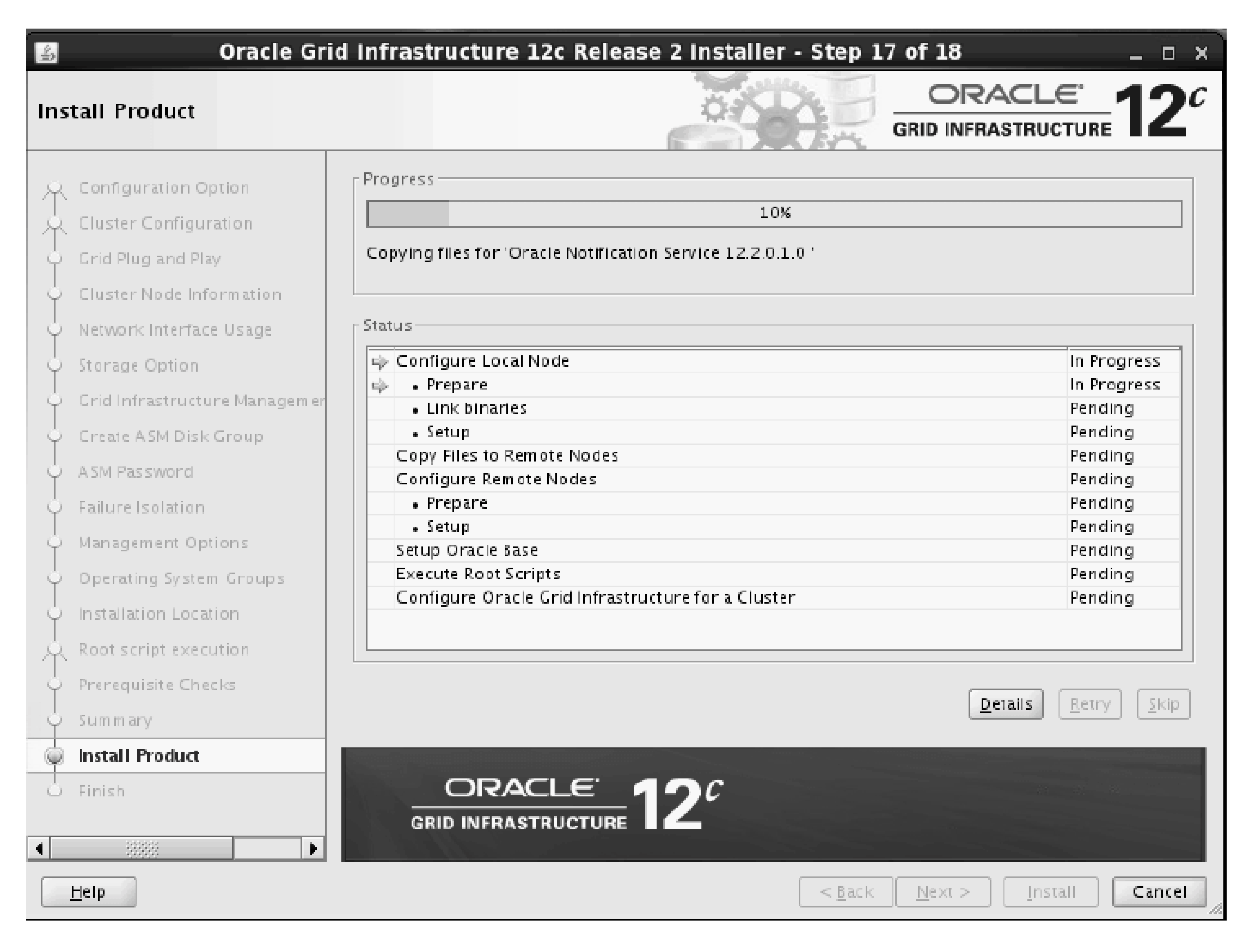


图 12-18　执行安装

17. 手动执行脚本

在安装的最后阶段，Oracle 需要人工执行两个脚本，请根据具体的提示进行。注意，需要使用 root 用户来执行。

```
orainstRoot.sh 脚本
[root@ptscsdb1grid]# /oracle/app/oraInventory/orainstRoot.sh
root.sh 脚本
```

```
[root@ptscsdb1grid]#  /oracle/app/12.2.0/grid/root.sh
```

18. 集群状态检查

安装完毕之后，检查集群状态是否正常，主要查看各种集群服务的状态。

```
[grid@ptscsdb1 ]$ crsctl stat res -t
```

12.5 创建 ASM 磁盘组

1. 启动 ASMCA

启动 ASM 的图形化管理工具，以便创建存放业务数据的磁盘组：

```
[grid@ptscsdb1]$ asmca
```

2. ASMCA 欢迎界面

如图 12-19 所示，启动 ASMCA，可以看到欢迎页面。请单击左侧目录中的 Disk Groups 选项。

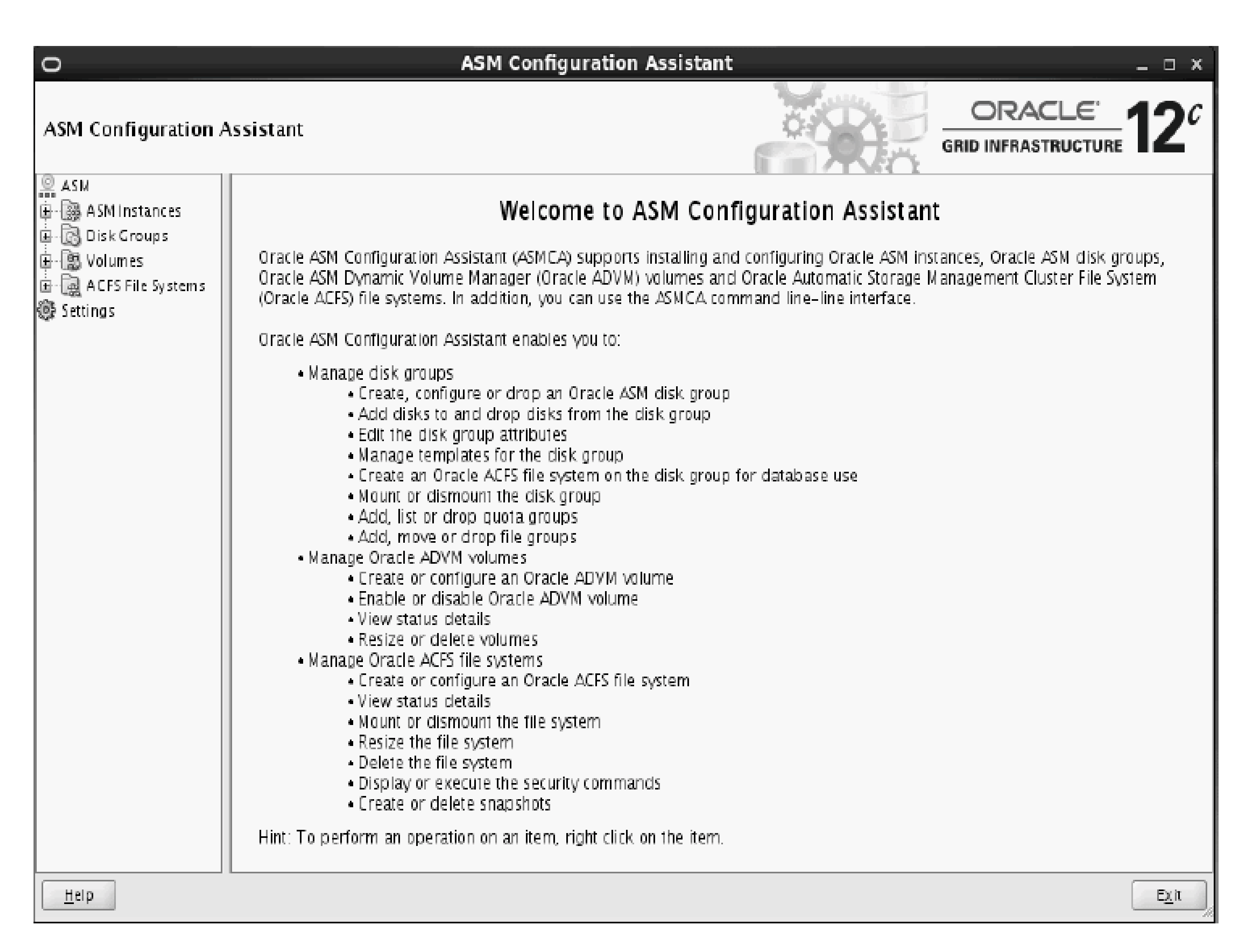


图 12-19　ASMCA 欢迎界面

3. 创建磁盘组

如图 12-20 所示，单击 Create 按钮开始创建新的磁盘组，存放业务数据。

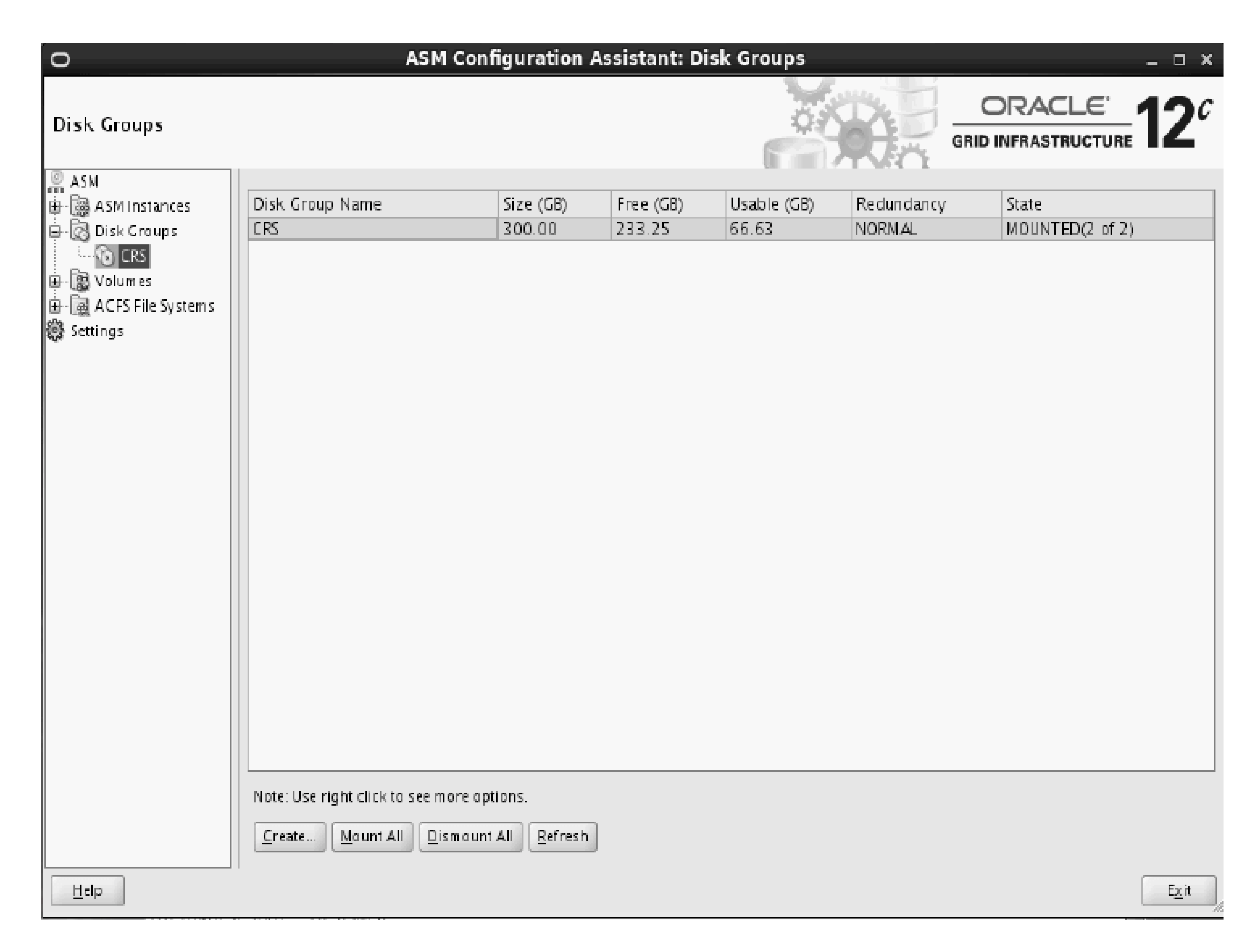


图 12-20　创建磁盘组

如图 12-21 所示，填写磁盘组的名称，同样没有固定的要求，请根据需要填写。冗余度建议选择外部（External），选择好磁盘组包含的磁盘之后，单击 OK 按钮开始创建。

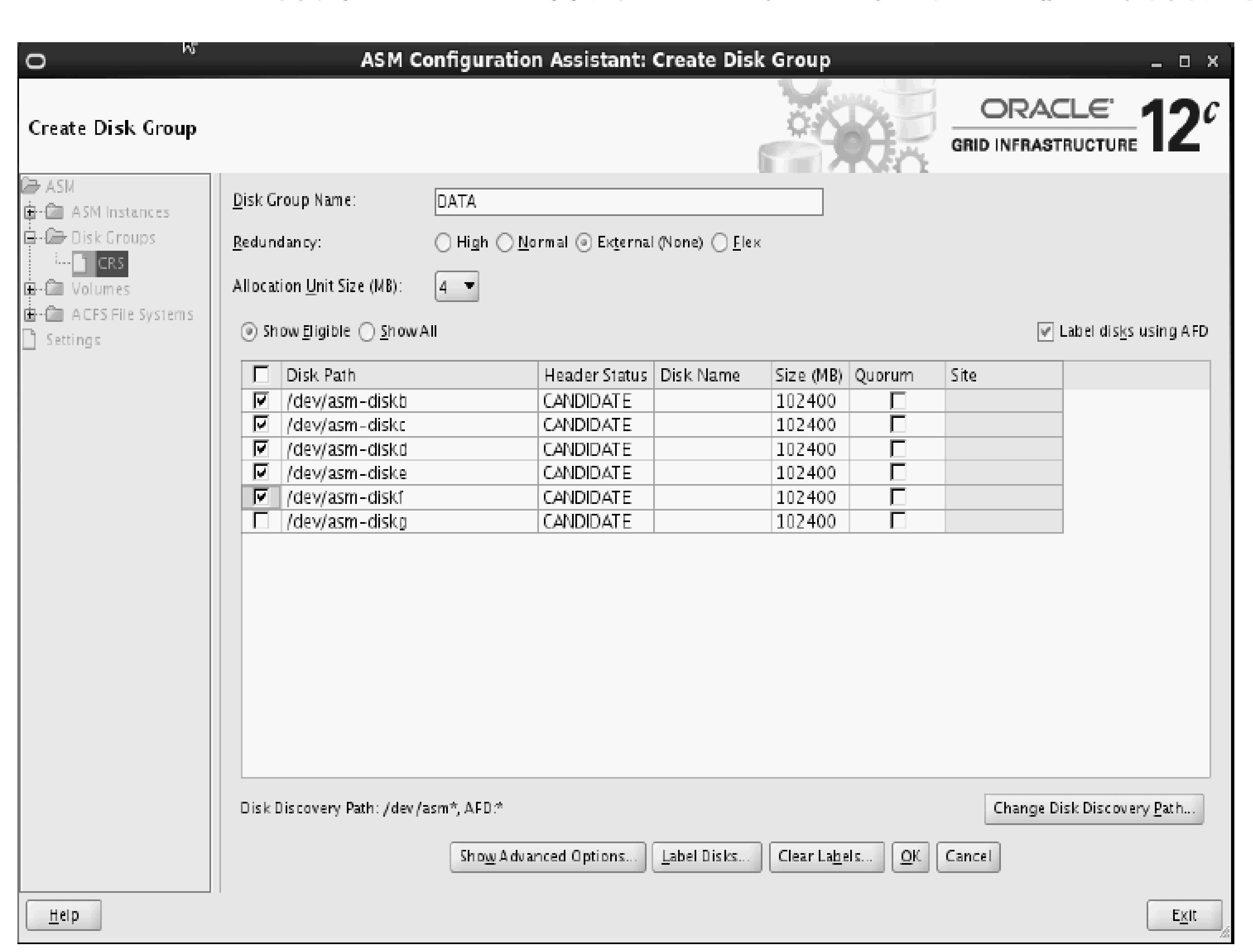


图 12-21　配置磁盘组

4. 磁盘组信息

创建完毕之后的磁盘组信息见图 12-22。

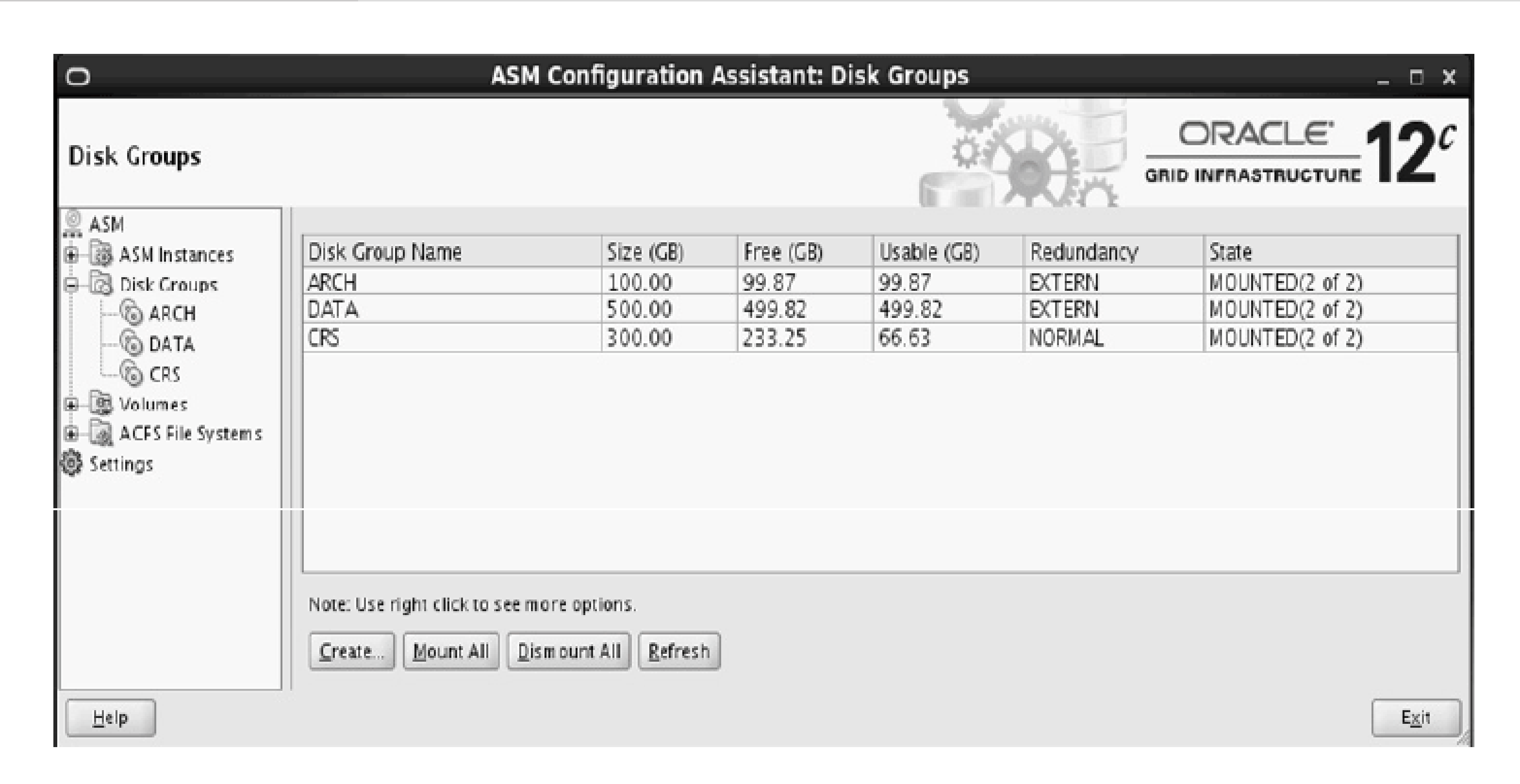


图 12-22　确认磁盘组信息

12.6 数据库软件安装

在集群软件安装完毕之后，开始安装数据库软件。请读者按照下面的步骤进行实践操作。

1. 开始安装

启动图形化安装程序，请使用 Oracle 用户执行：

```
[oracle@ptscsdb1]&./runInstaller
```

2. 配置 support 用户（不需要配置）

如图 12-23 所示，请不要选择任何选项，单击 Next 按钮。

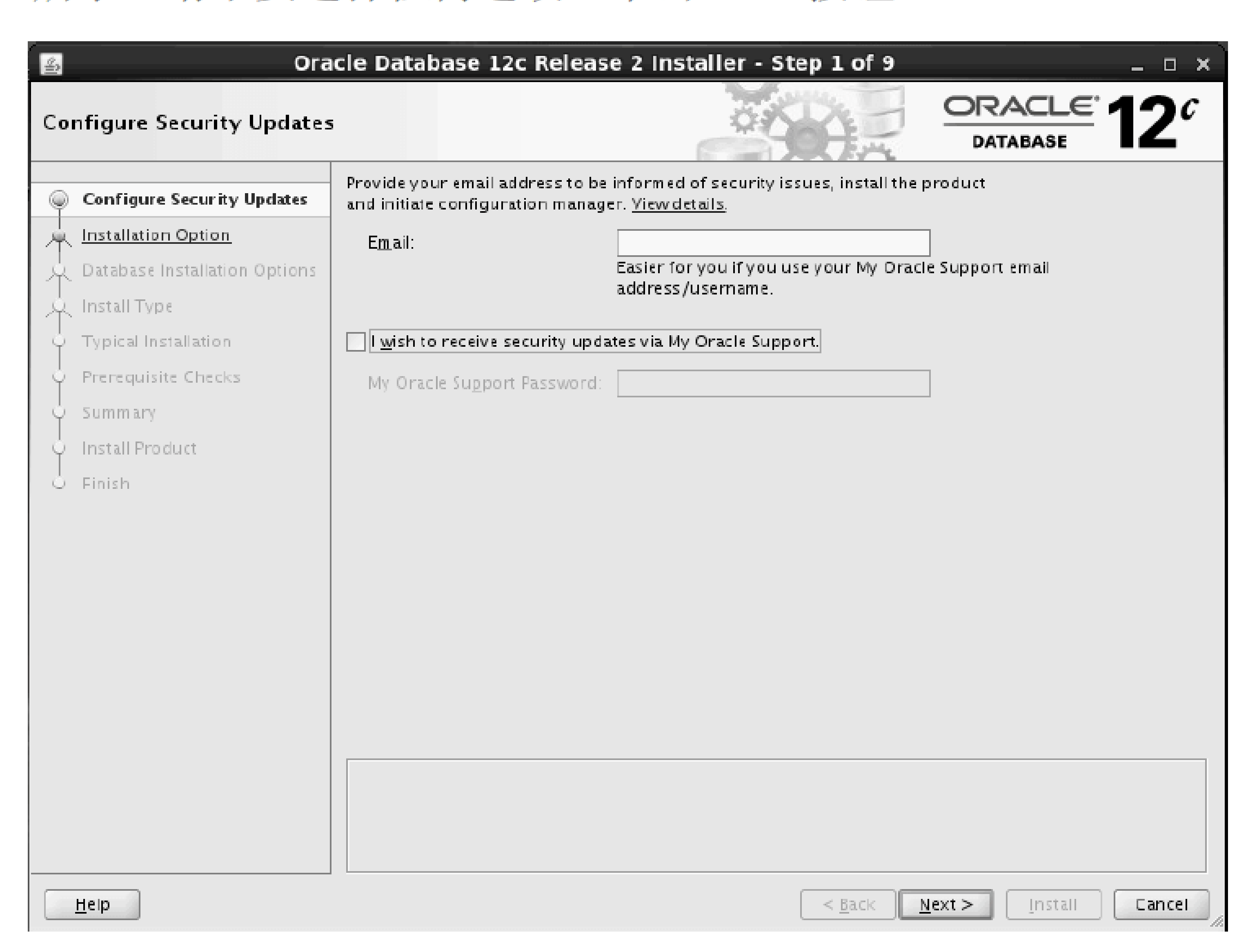


图 12-23　Oracle support 配置

3. 选择安装选项

如图 12-24 所示，请选择第二项，即只安装数据库软件选项，单击 Next 按钮。

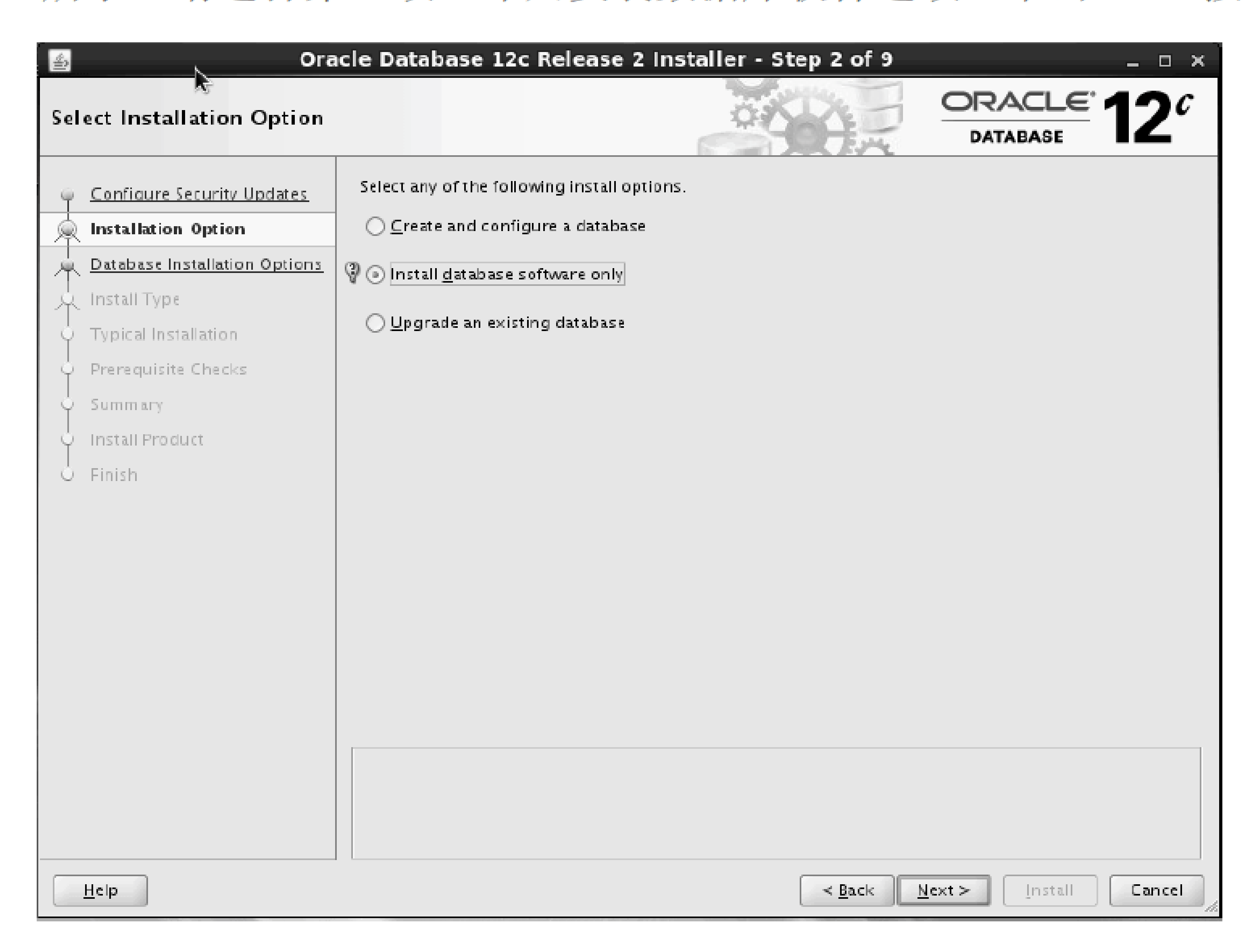


图 12-24　选择安装选项

4. 选择数据库类型

如图 12-25 所示，请选择第二项，安装高可用集群模式的数据库软件安装，单击 Next 按钮。

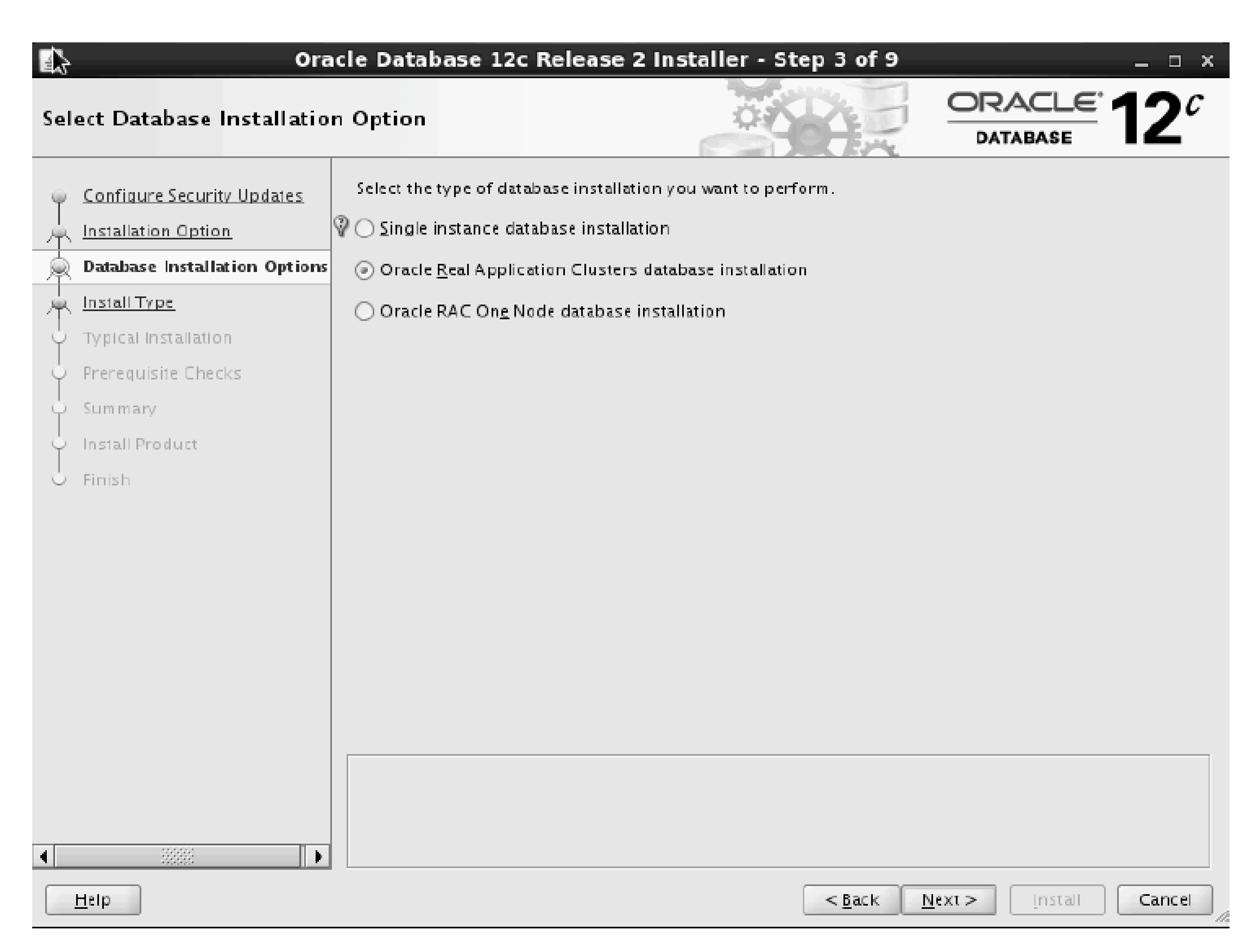


图 12-25　选择数据库类型

5. 选择安装节点配置 SSH 连通性

如图 12-26 所示，将两个节点的主机全部勾选，因为之前已经进行了互信的配置，这里的互信配置客户忽略，单击 Next 按钮。

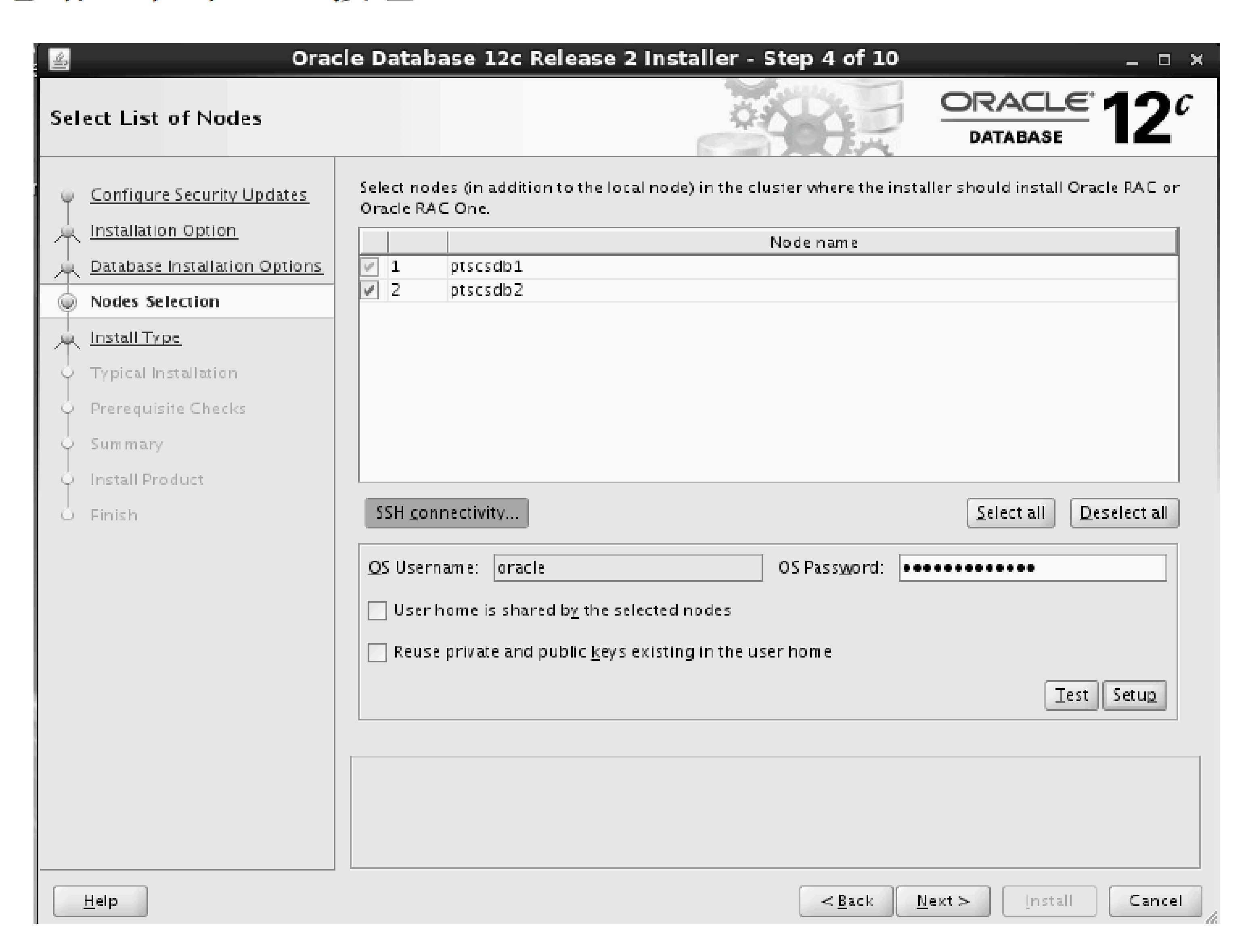


图 12-26　节点与互信

6. 选择数据库版本

如图 12-27 所示，选择第一项，安装企业版的数据库软件，单击 Next 按钮。



图 12-27　数据库版本

7. 指定安装目录

如图 12-28 所示，填写数据库软件的安装位置，如果 Oracle 用户环境变量设置正确，安装程序可以自动填写目录，单击 Next 按钮。

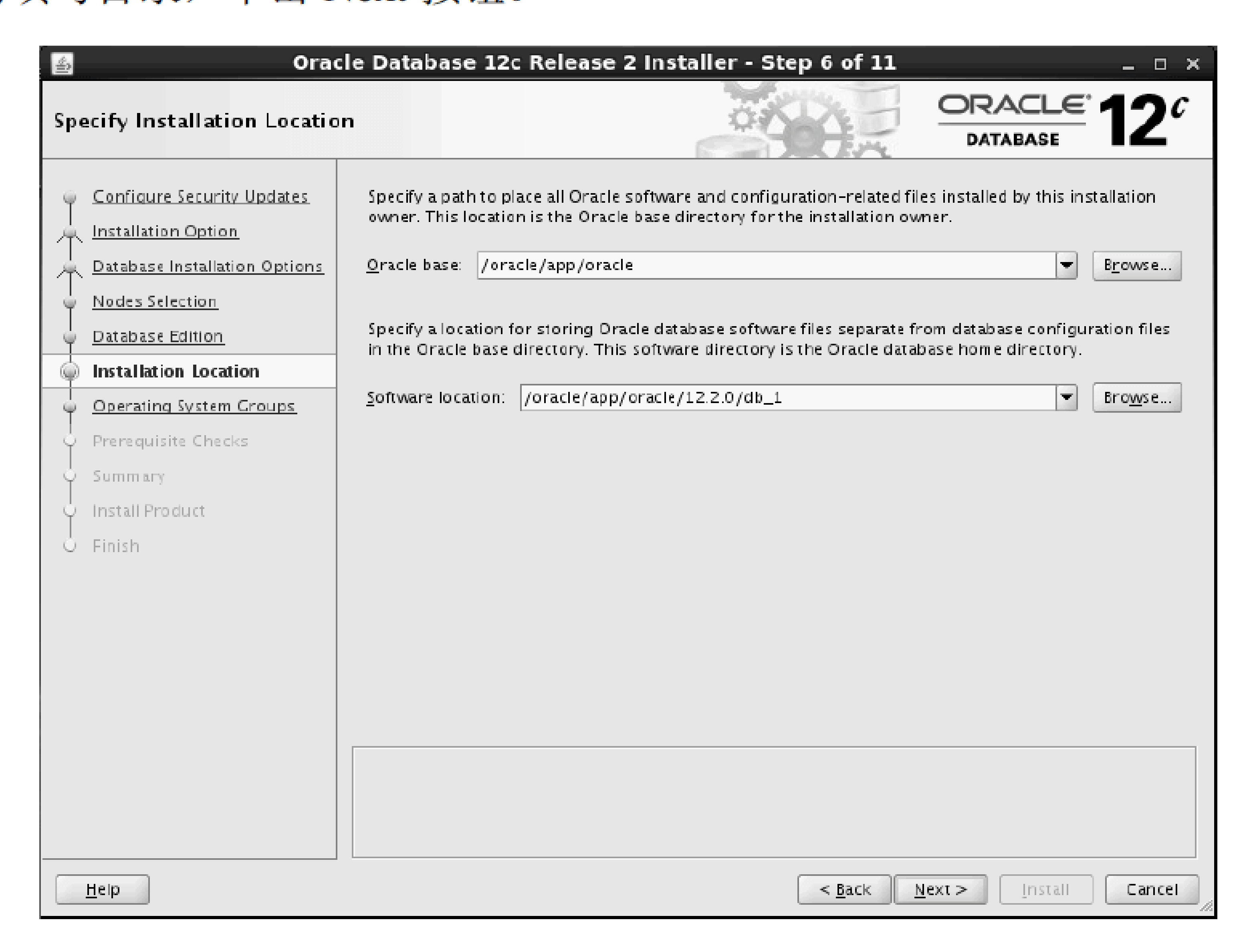


图 12-28　安装目录

8. 选择用户组

如图 12-29 所示，选好全部的用户组，单击 Next 按钮。

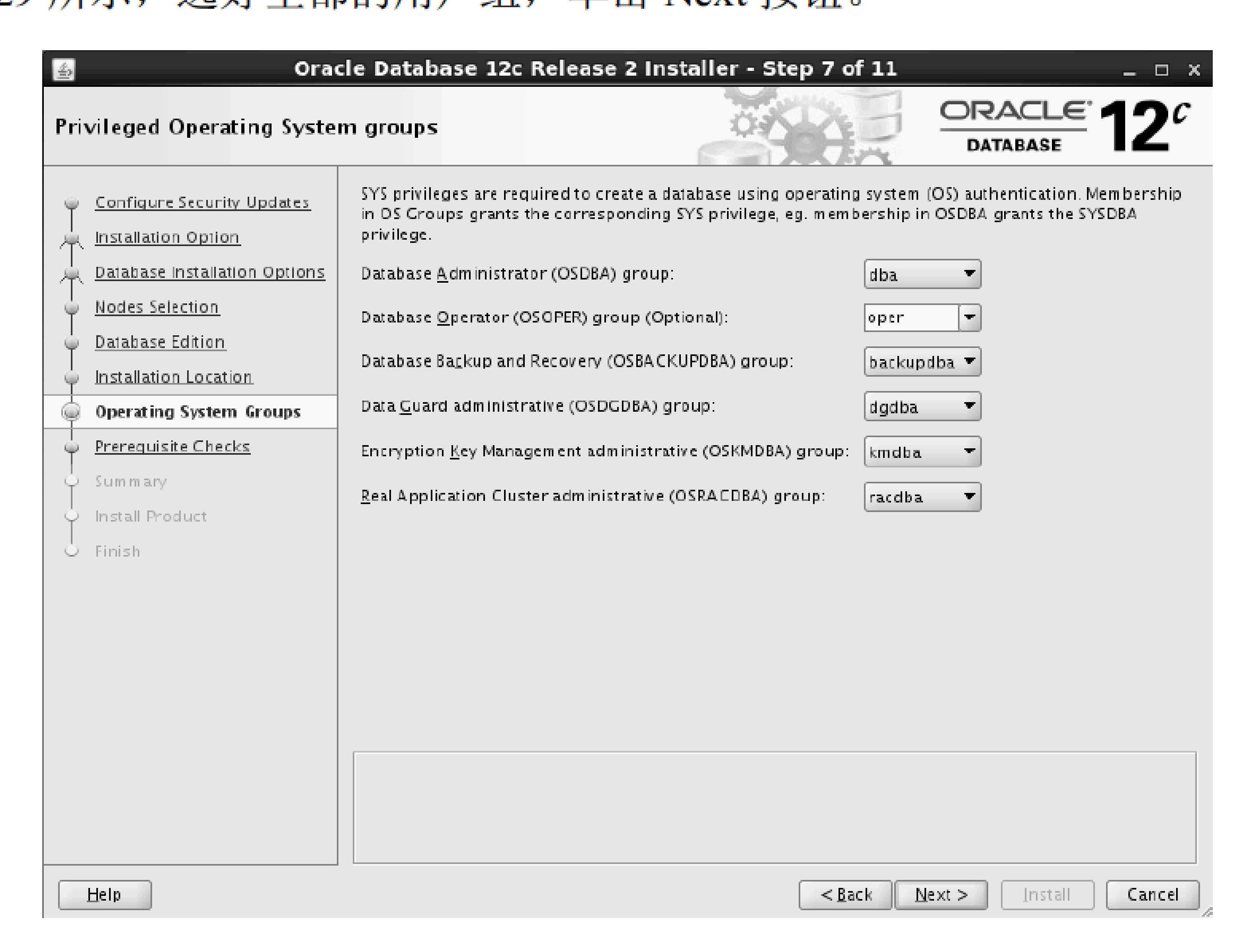


图 12-29　用户和组

9. 安装预检查

如图 12-30 所示，安装程序开始进行最后的环境检查。

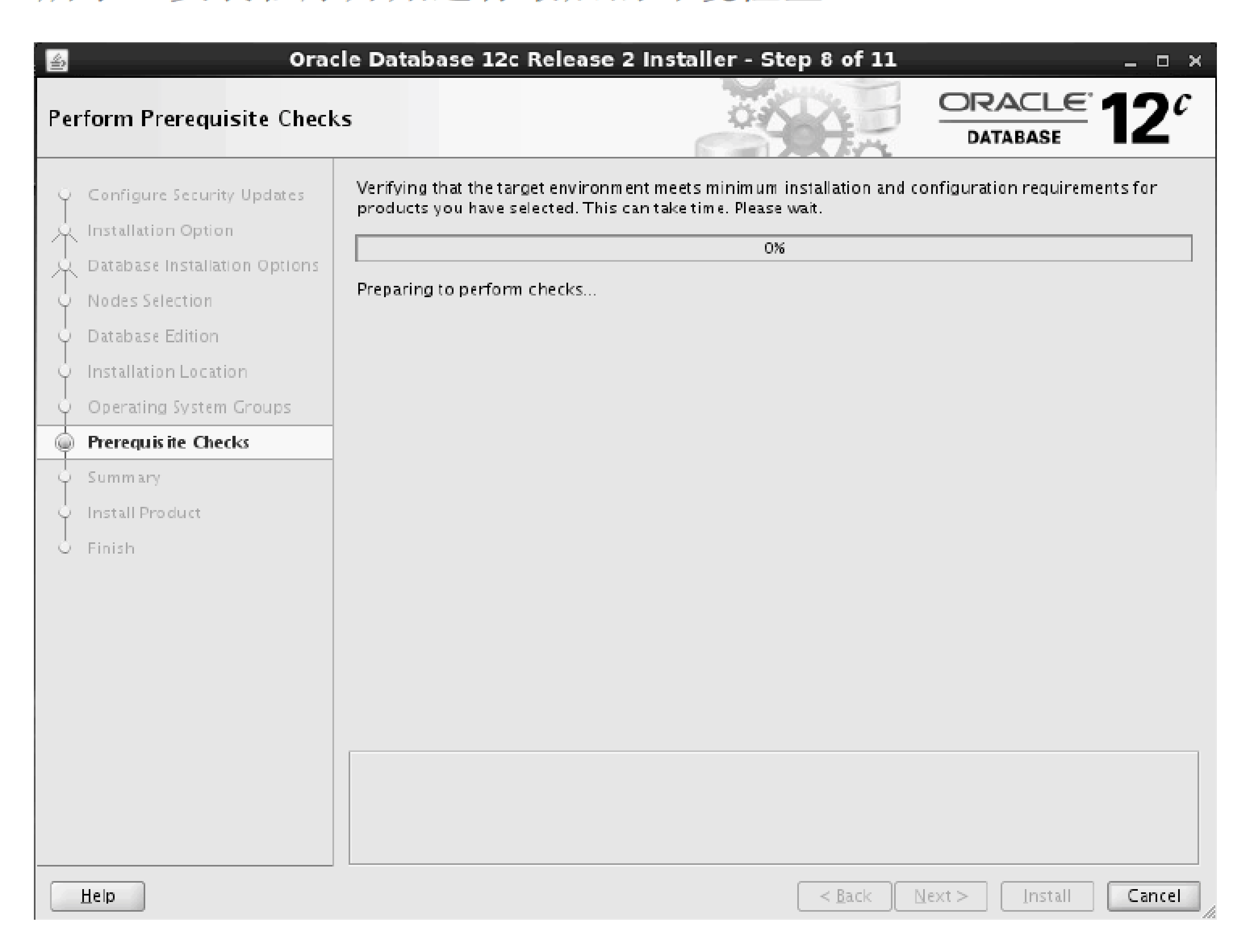


图 12-30　安装预检查

如图 12-31 所示，请勾选右上角的 Ignore ALL，因为所列出的选项对于软件安装和使用的影响基本可以忽略，然后单击 Next 按钮。

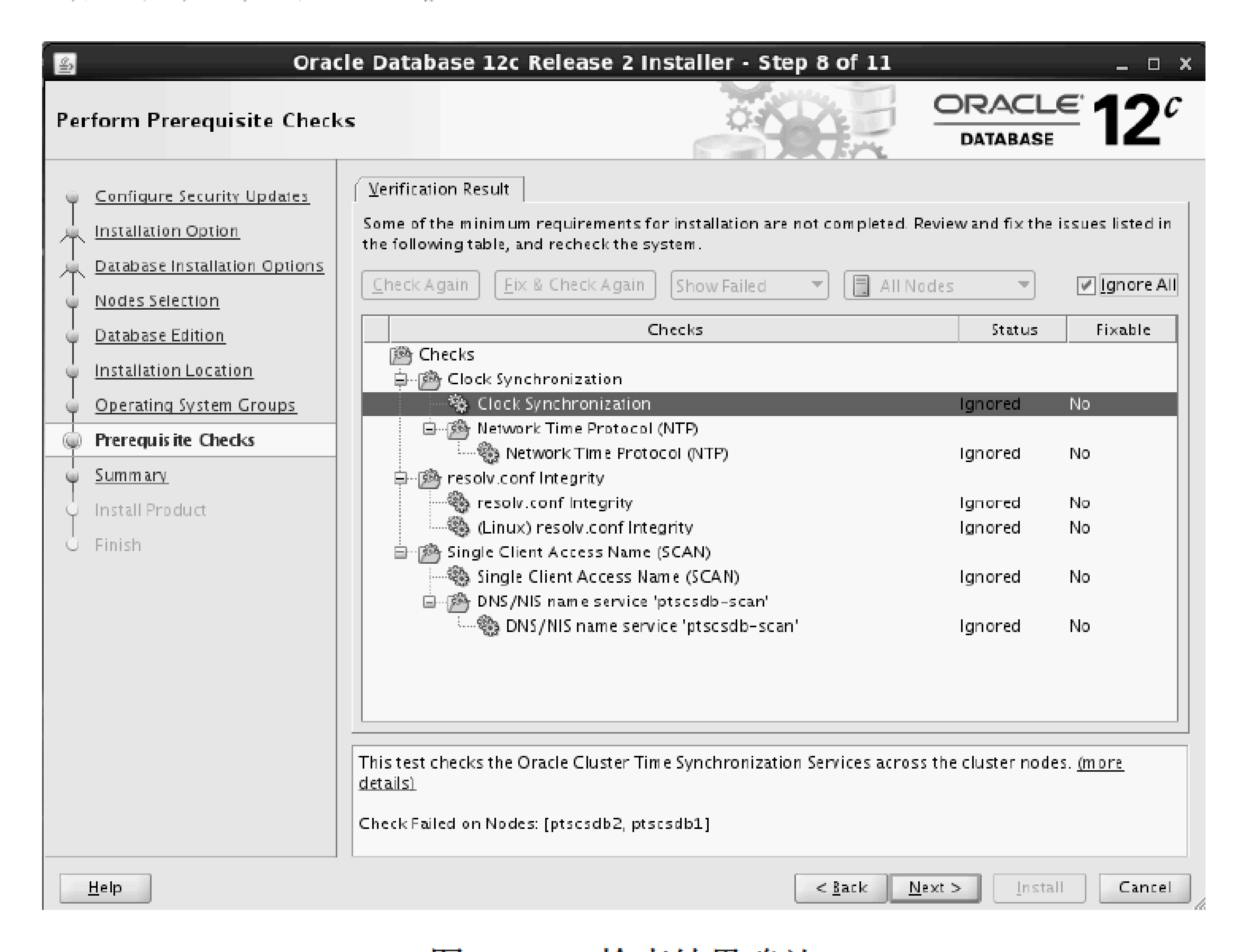


图 12-31　检查结果确认

10. 安装概要

如图 12-32 所示，单击 Install 按钮开始正式安装。

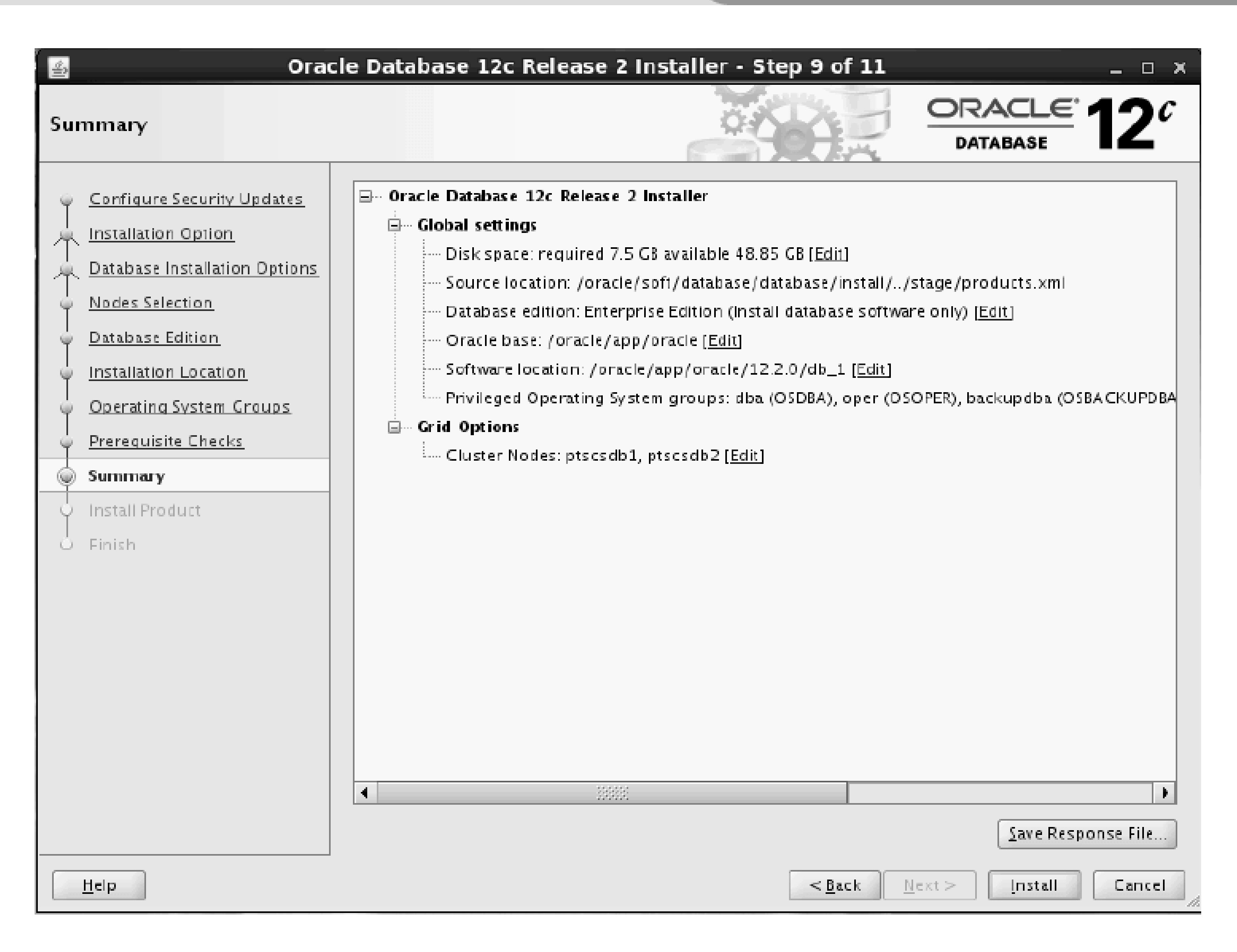


图 12-32　安装概要

11. 执行安装

如图 12-33 所示，程序开始执行安装，并显示安装进度和具体的安装选项。

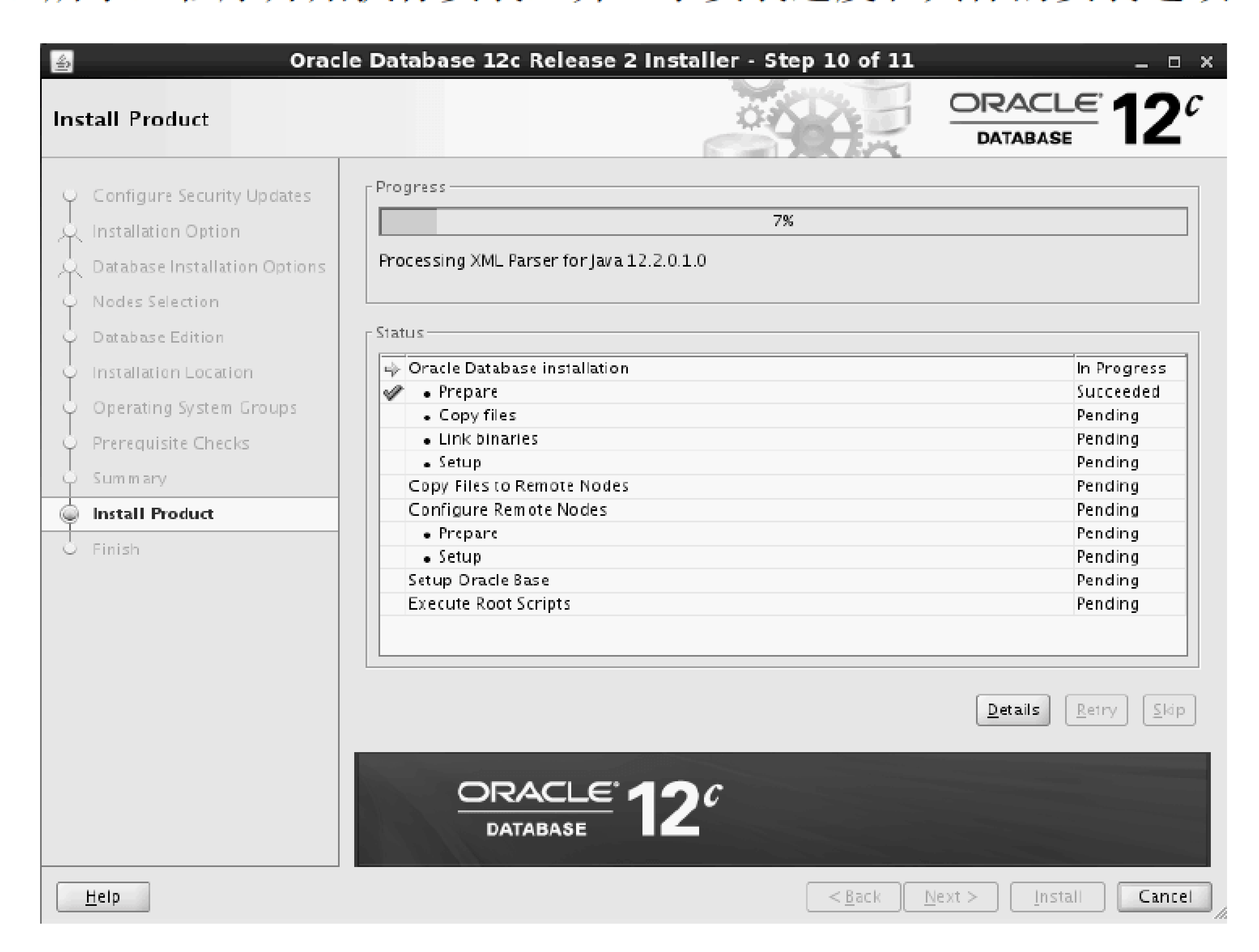


图 12-33　安装过程

12. 执行脚本

如图 12-34 所示，在程序安装的最后阶段，Oracle 会提示请用户使用 root 用户执行脚本。

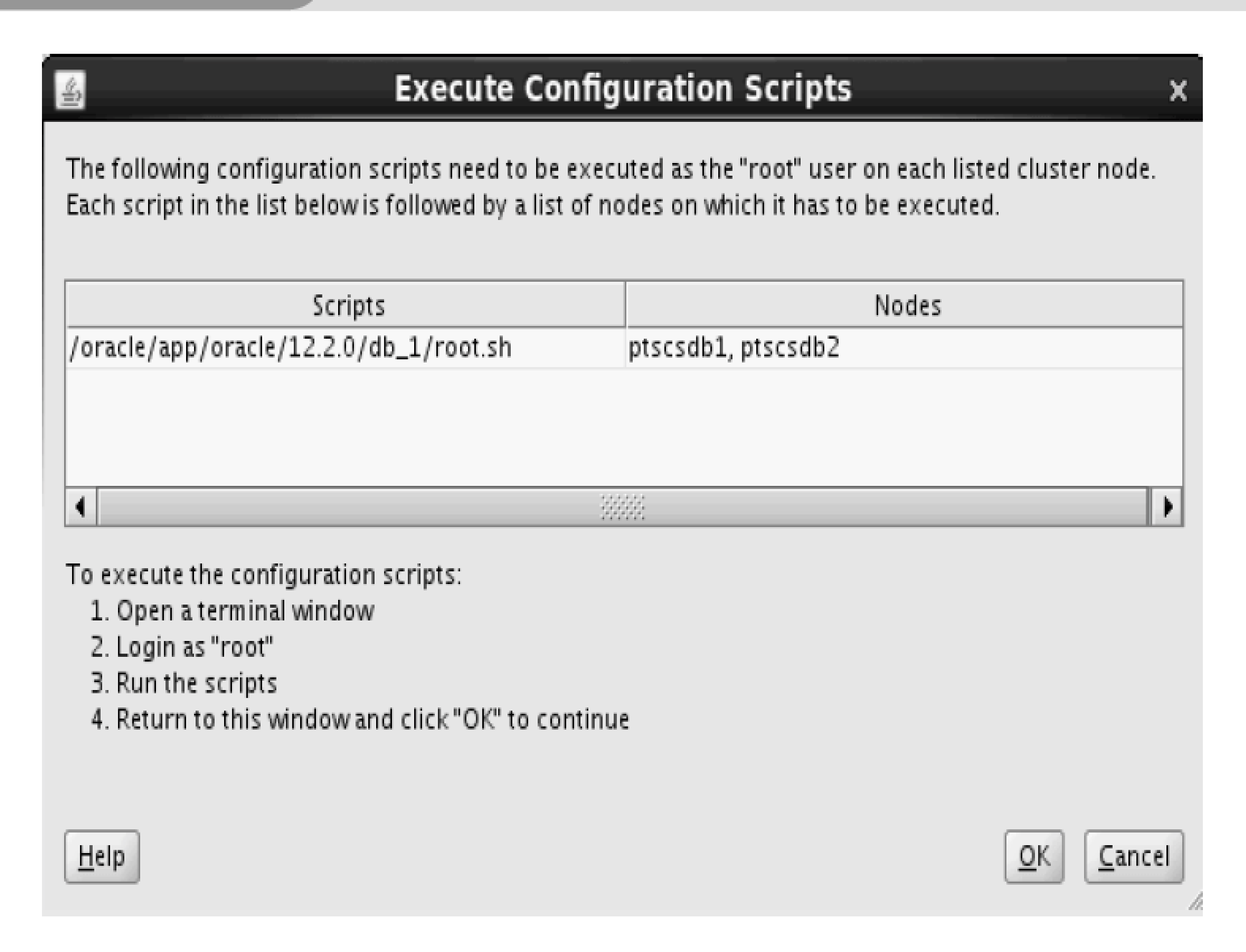


图 12-34　root 脚本执行

13. 安装完成

如图 12-35 所示，数据库软件的安装到此结束，单击 Close 按钮完成。

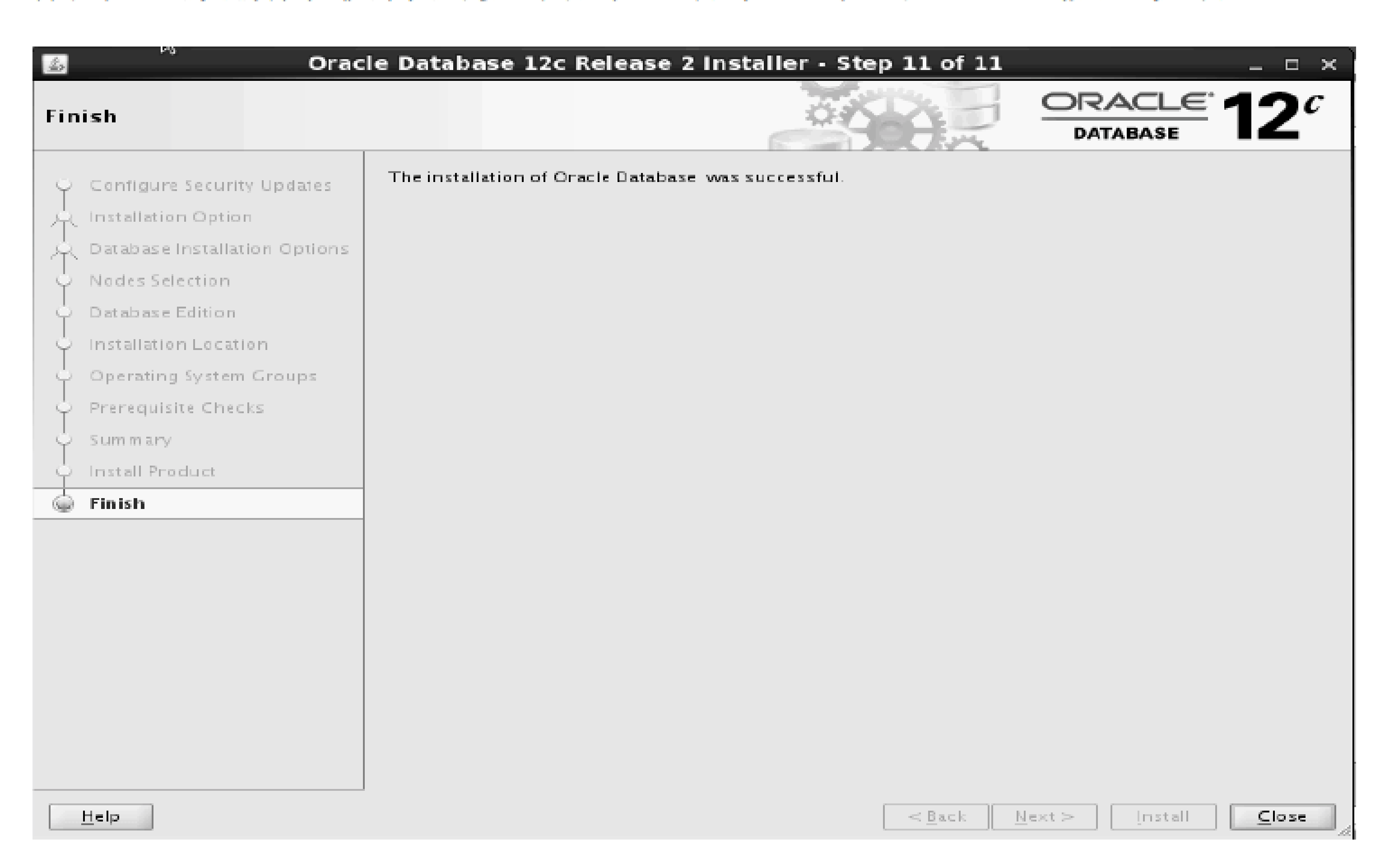


图 12-35　安装结束

12.7 建库

在集群软件和数据库软件安装完毕之后，基础的环境已经准备就绪，这时其实还没有数据库实例。数据库实例需要通过 DBCA 工具进行创建，请读者按照如下方法进行练习实践。

1. 执行 DBCA

需要使用 Oracle 用户启动 DBCA：

```
[oracle@ptscsdb1]&dbca
```

2. 选择创建数据库

如图 12-36 所示，选择第一项创建数据库实例，单击 Next 按钮。

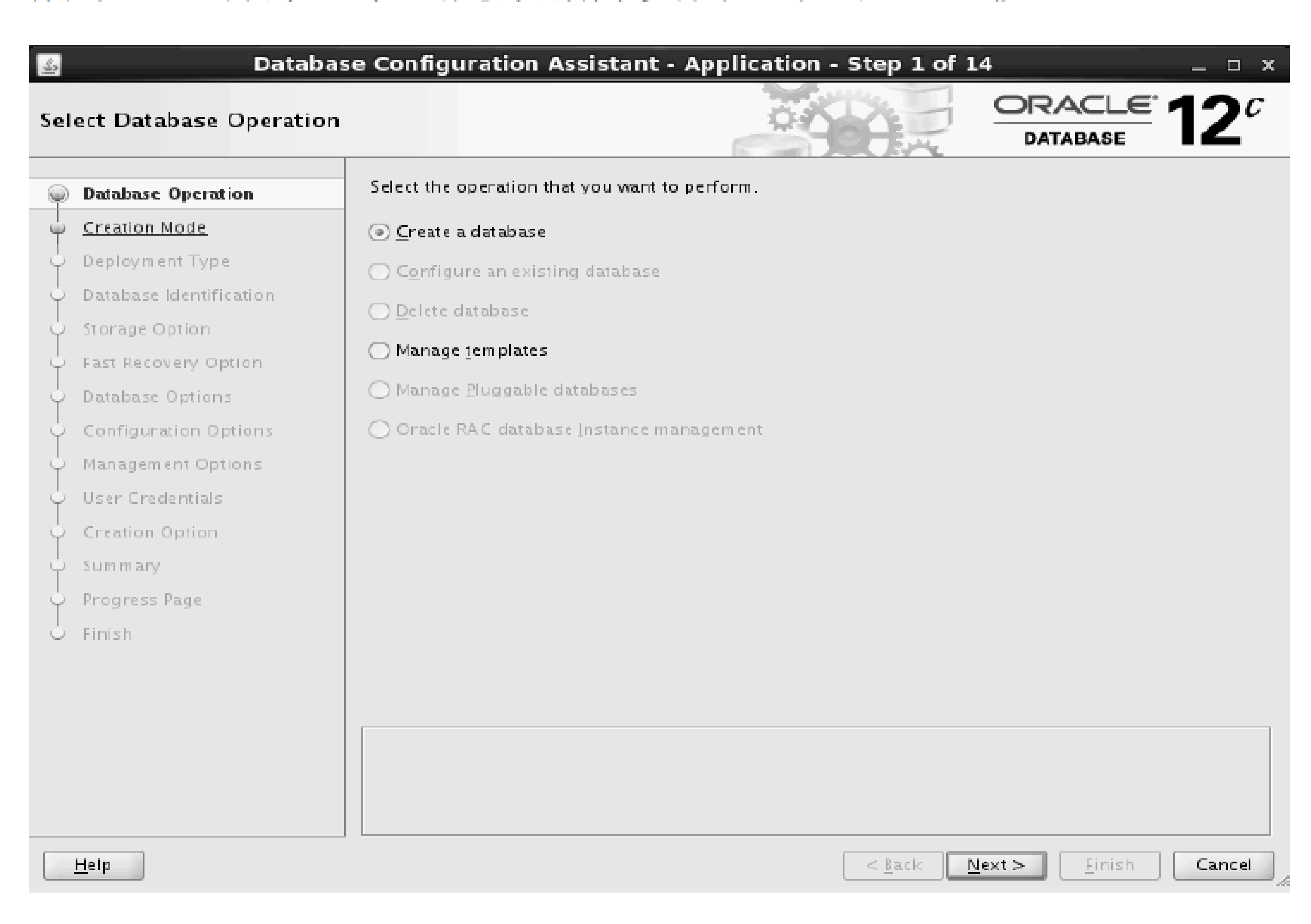


图 12-36 创建数据库实例

3. 选择创建模式

如图 12-37 所示，选择高级选项安装，单击 Next 按钮。

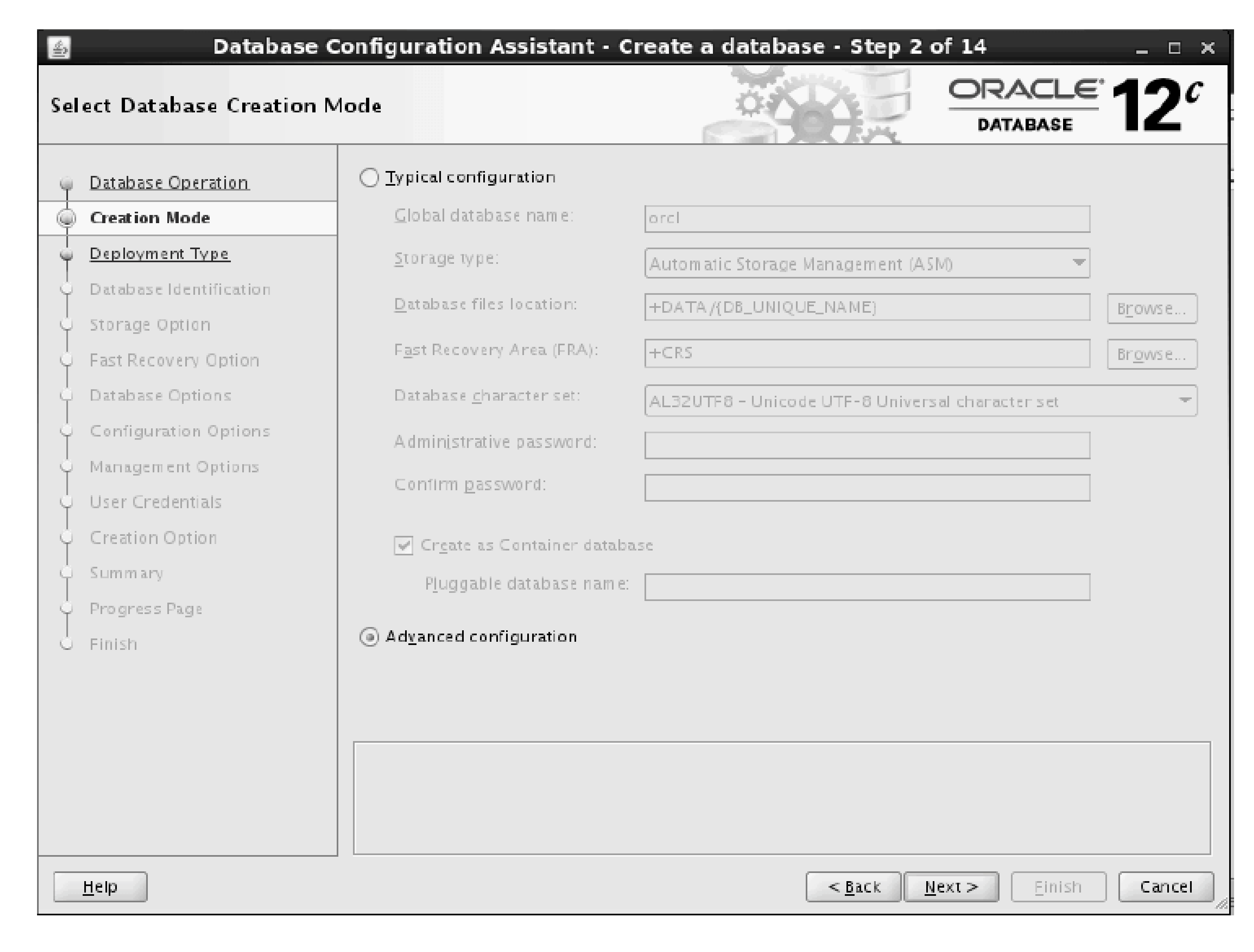


图 12-37 选择创建模式

4. 选择数据库模板

如图 12-38 所示，数据库的类型选择 Oracle RAC，配置类型选择 Admin 的管理方式，数据库实例的安装类型选择第二项，一般或交易型数据库，这也是大多数生产环境的类型，单击 Next 按钮。

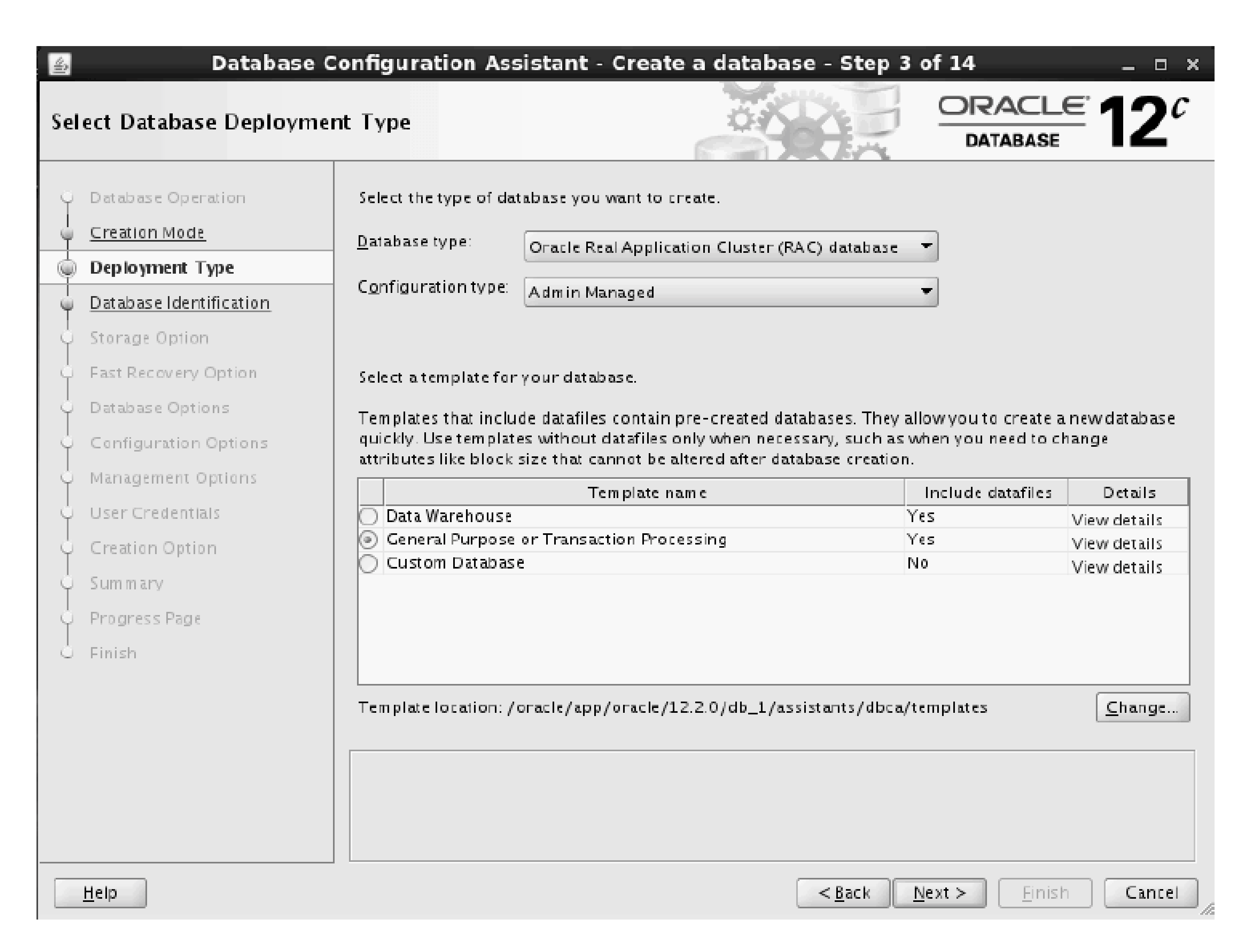


图 12-38　选择数据库模板

5. 选择安装节点

如图 12-39 所示，请将所有节点进行勾选，单击 Next 按钮。

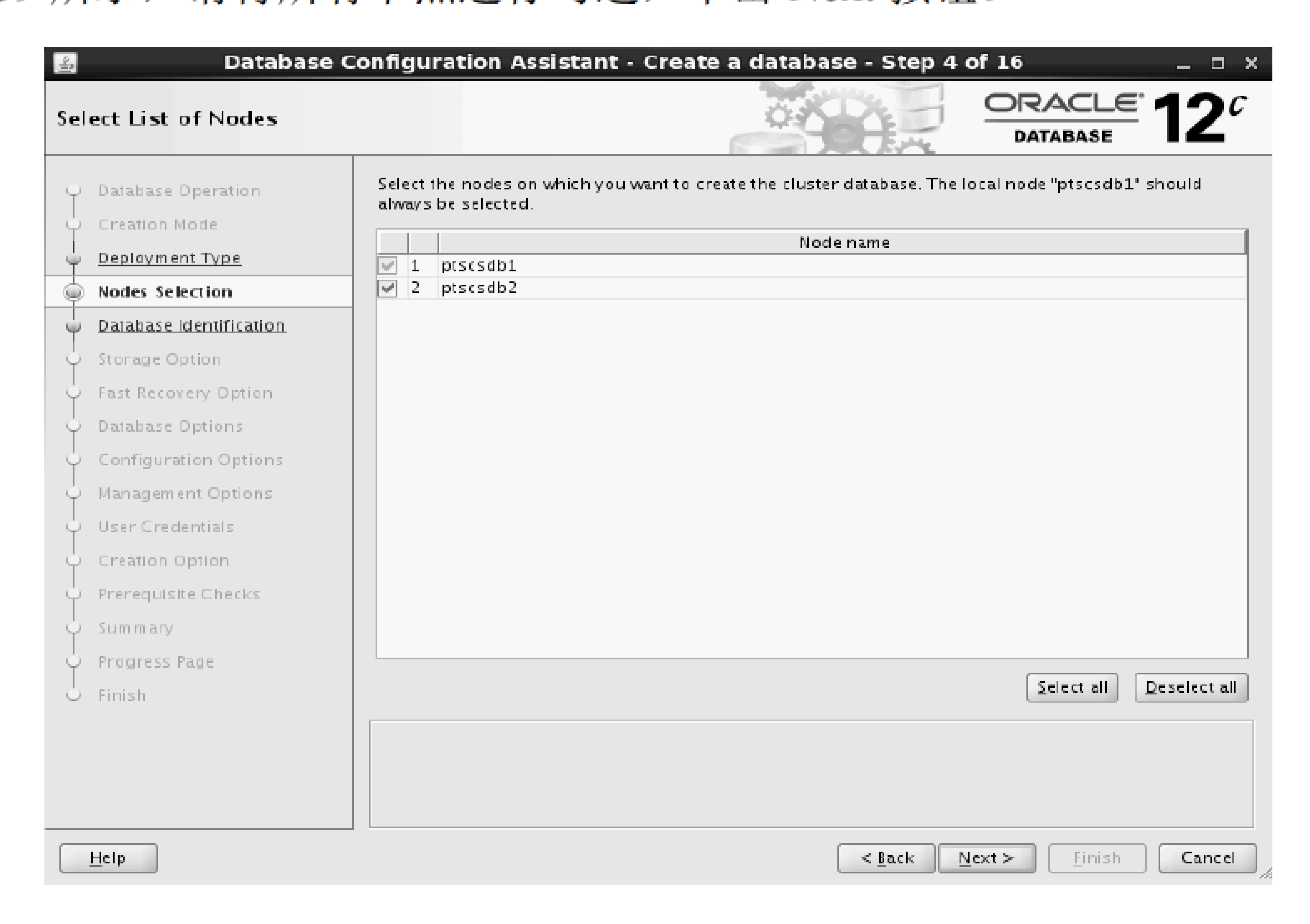


图 12-39　选择安装节点

6. 指定数据库信息

如图 12-40 所示，填写全局的数据库实例名字，即数据库名称，SID Prefix 名称与数据库名称保持一致，单击 Next 按钮，其他选项请不要勾选。

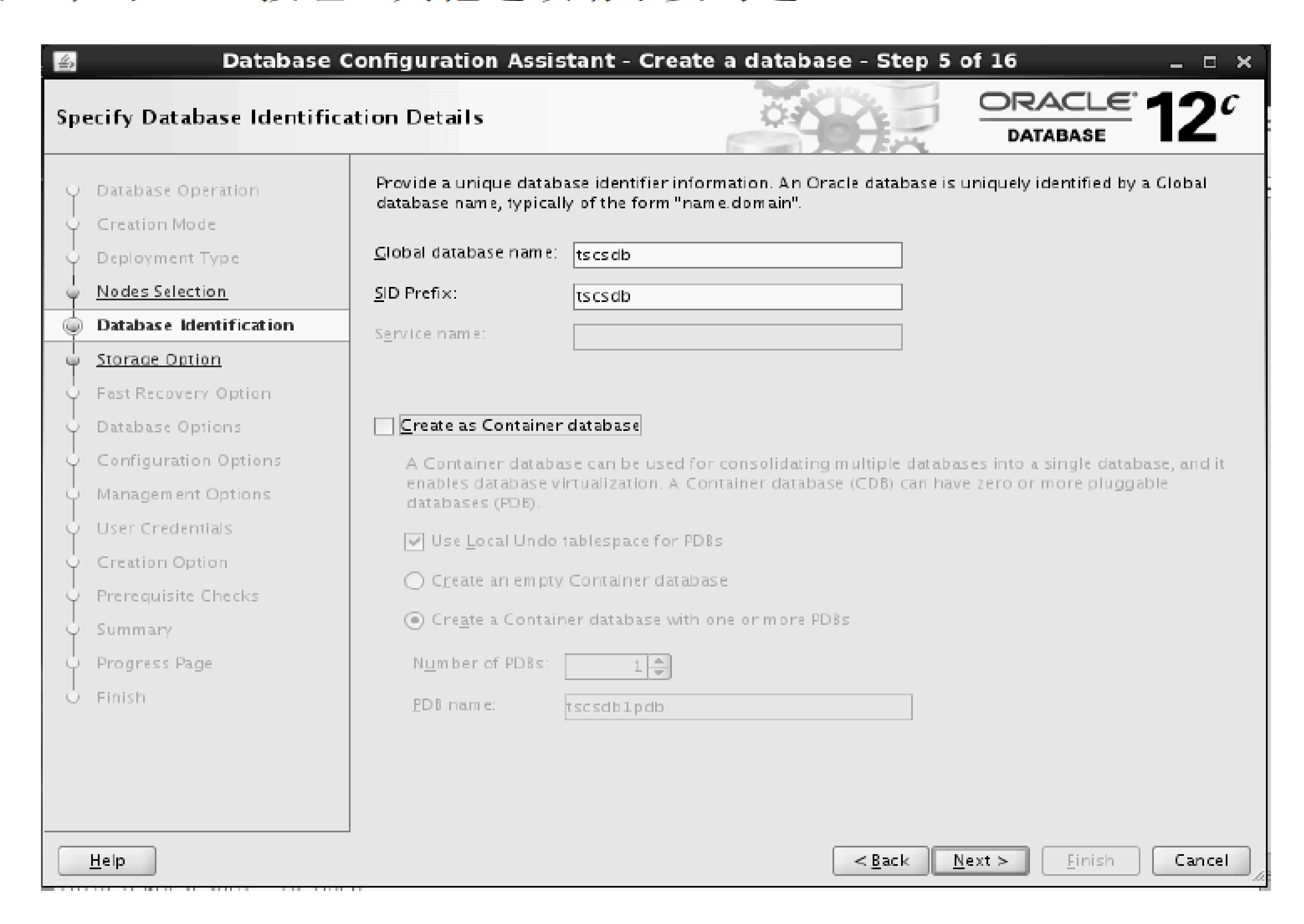


图 12-40　指定数据库信息

7. 选择存储选项

如图 12-41 所示，选择数据存放的位置，请选择之前创建好的 ASM 磁盘组，存放的类型为 ASM，单击 Next 按钮，其他选项请不要勾选。

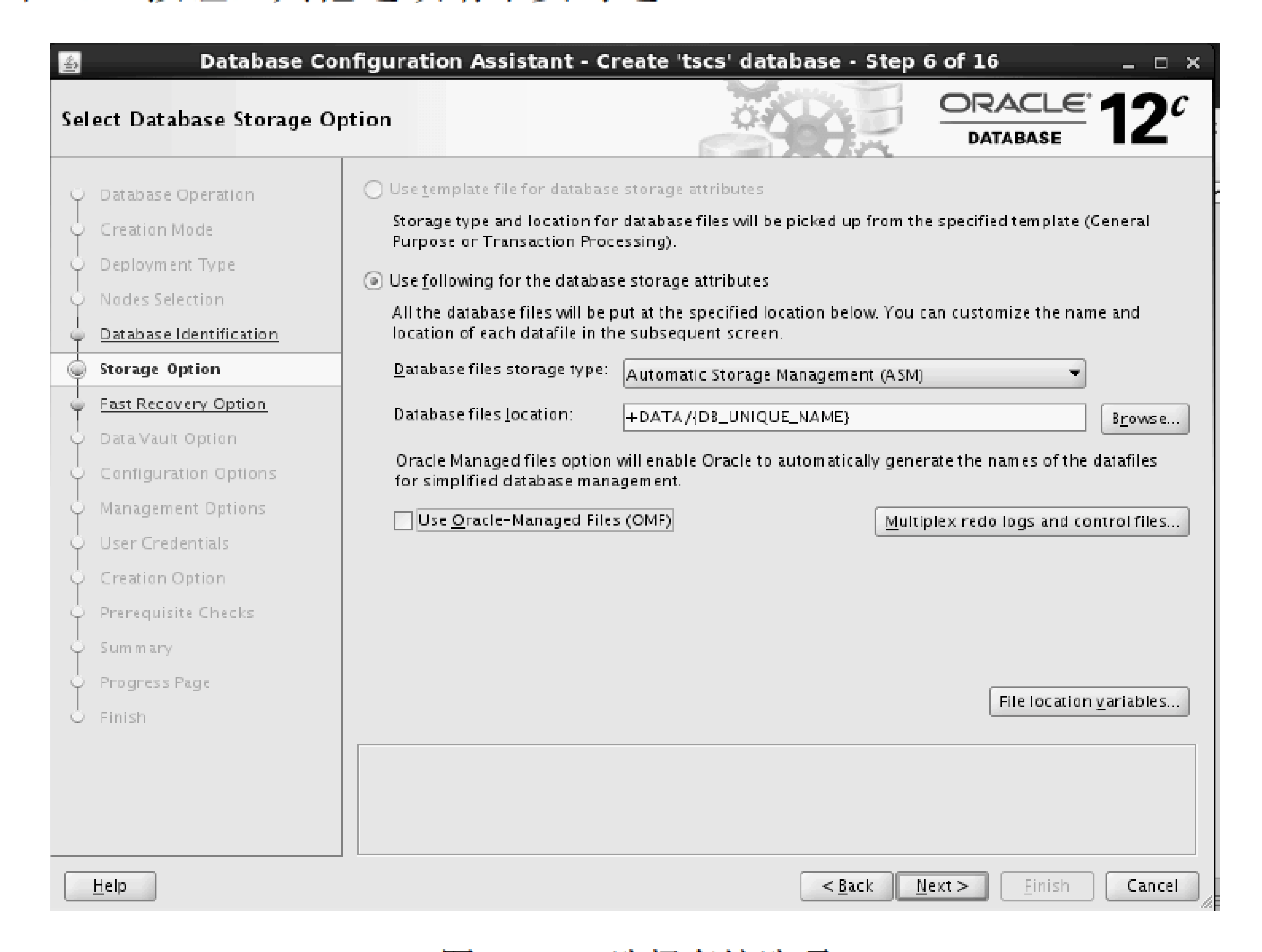


图 12-41　选择存储选项

8. 快速恢复区选项

如图 12-42 所示，该步骤需要设置存放归档文件的空间，请勾选启用，单击 Next 按钮。

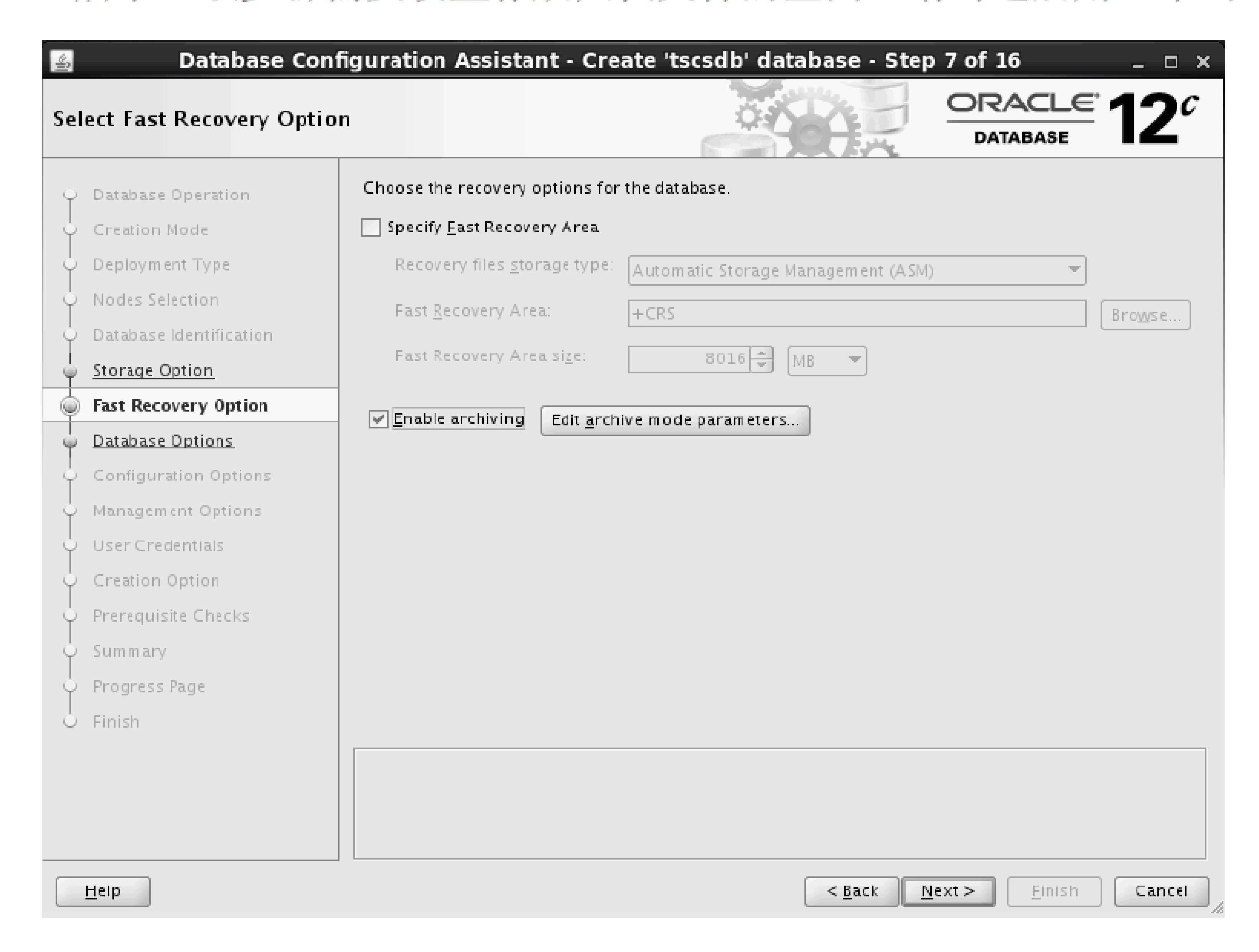


图 12-42　归档空间选项

9. 数据资料库选项

如图 12-43 所示，请不要选择任何选项，单击 Next 按钮。

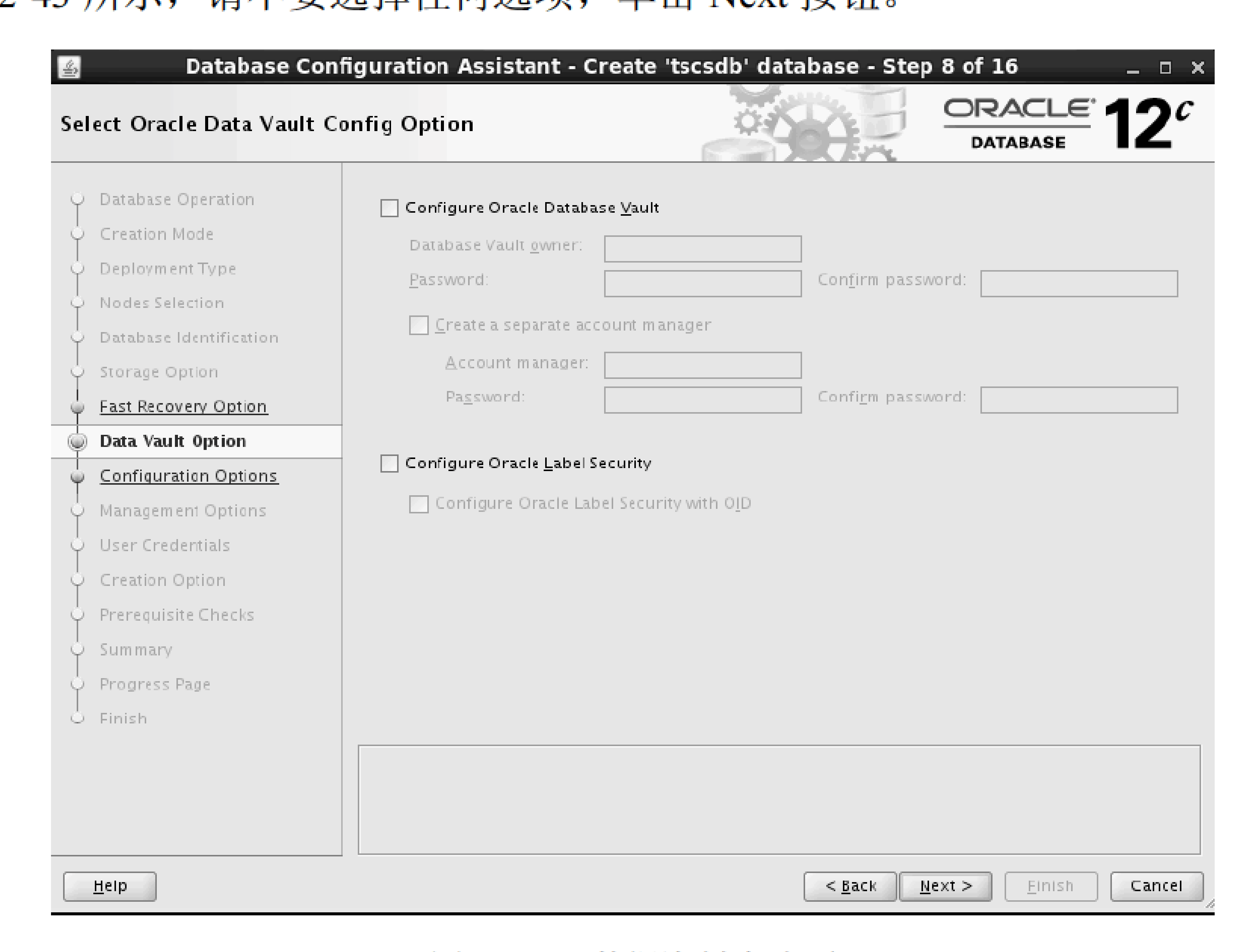


图 12-43　数据资料库选项

10. 配置数据库参数

如图 12-44 所示，保持默认选项即可，单击 Next 按钮。

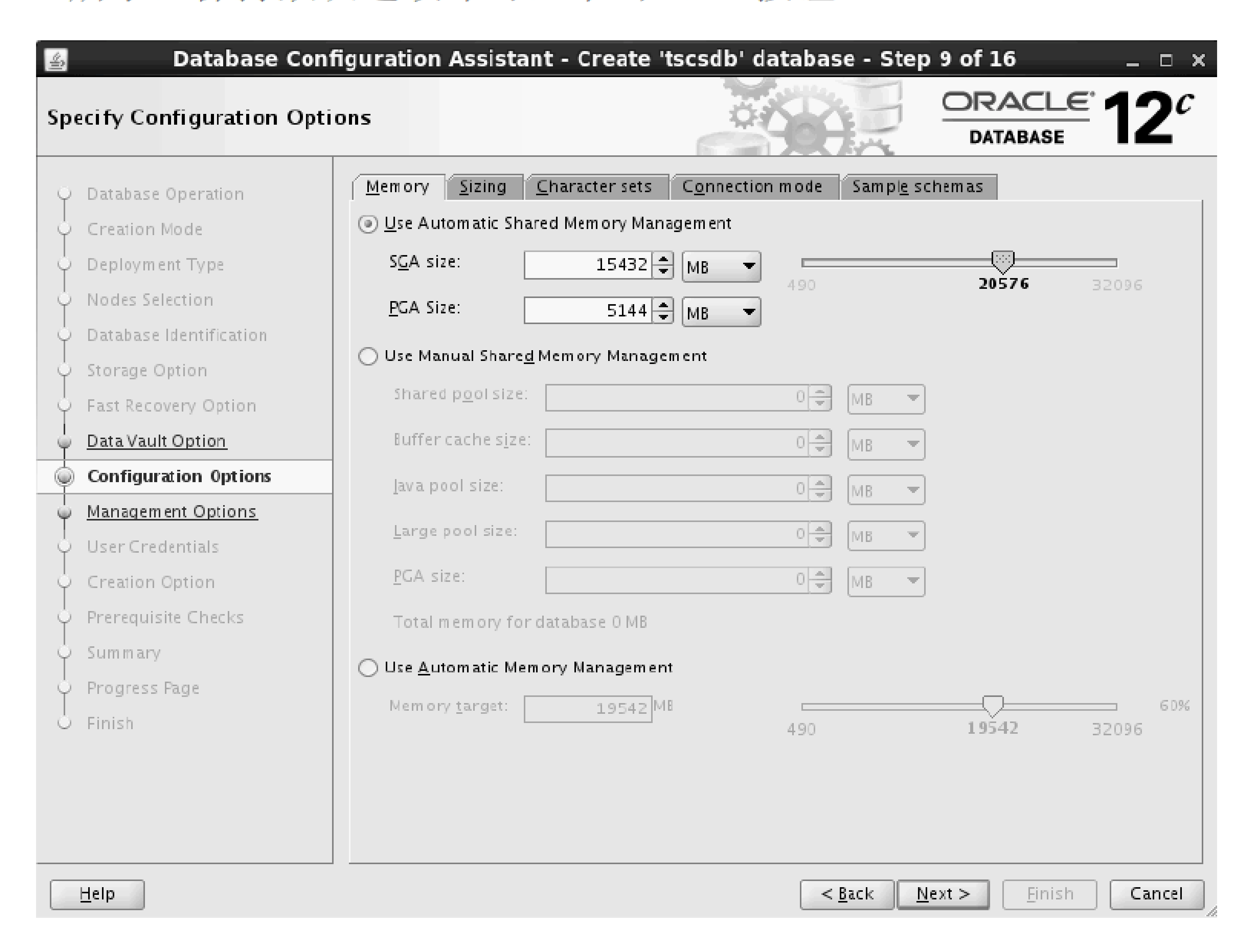


图 12-44　数据库参数配置

11. 指定管理选项

如图 12-45 所示，请不要选择任何选项，单击 Next 按钮。

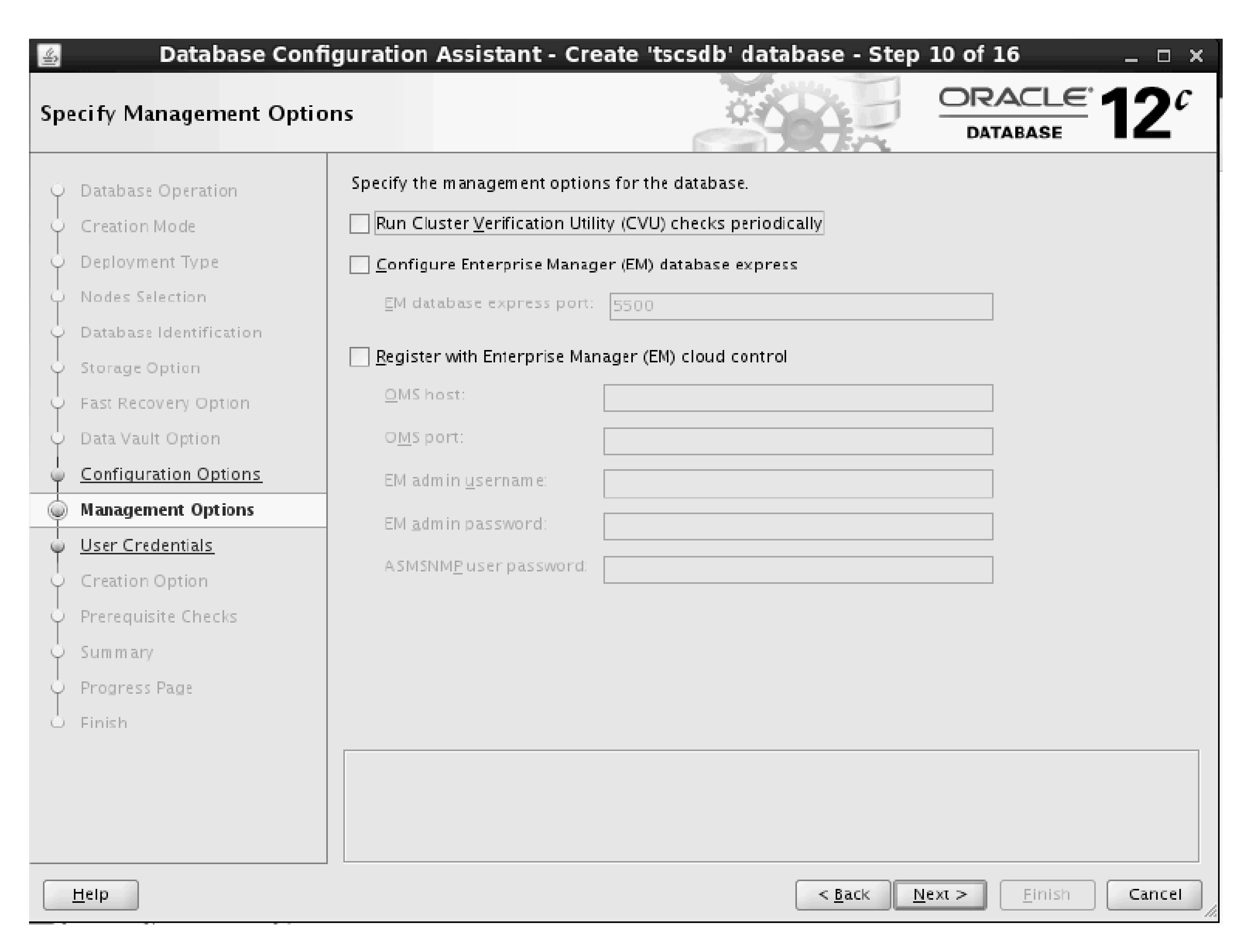


图 12-45　管理选项

12. 指定数据库用户密码

如图 12-46 所示，为数据库管理员设置密码之后，单击 Next 按钮。

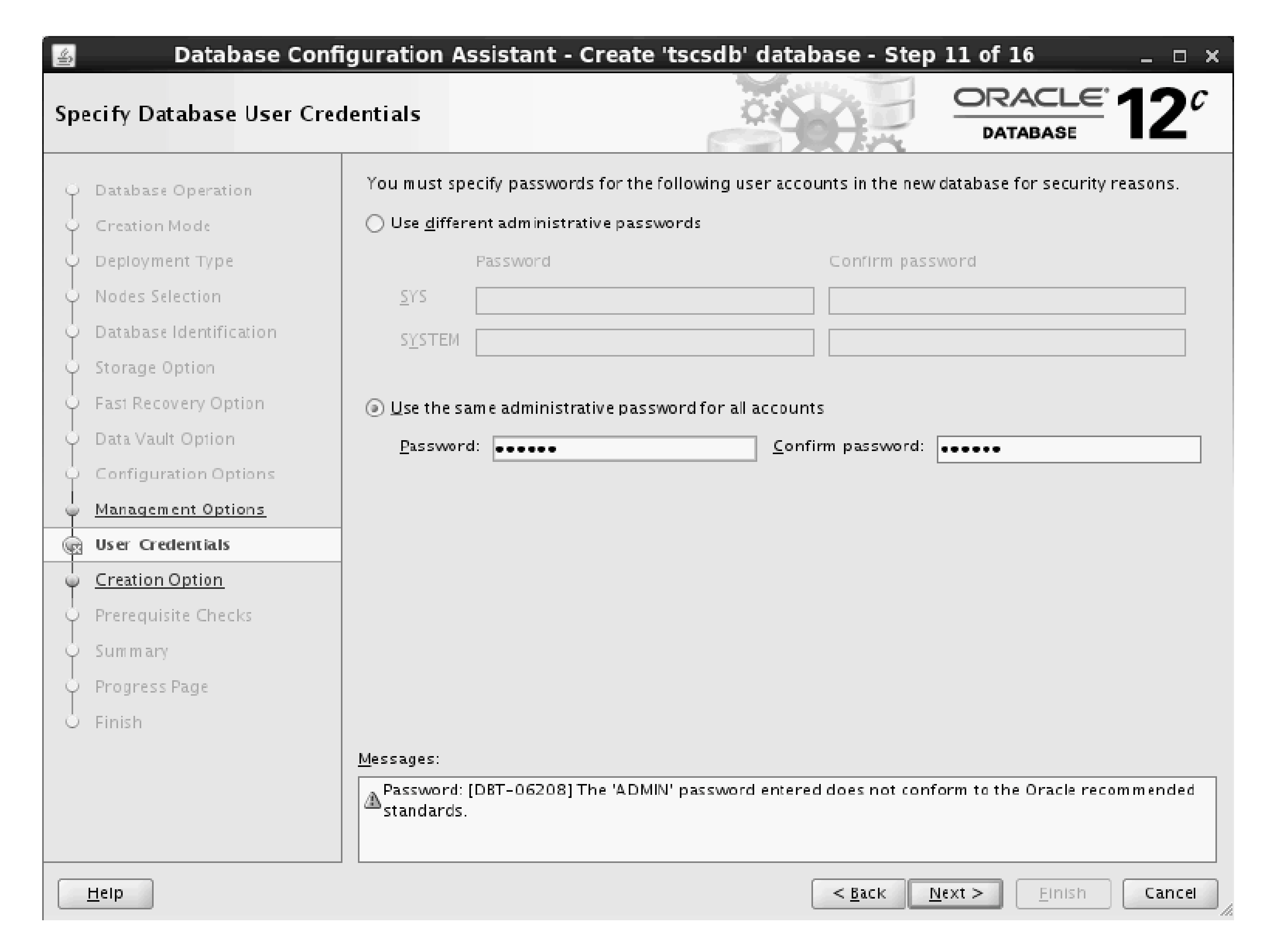


图 12-46　数据库实例密码

13. 选择数据库创建选项

如图 12-47 所示，请勾选第一项创建数据库，单击 Next 按钮。

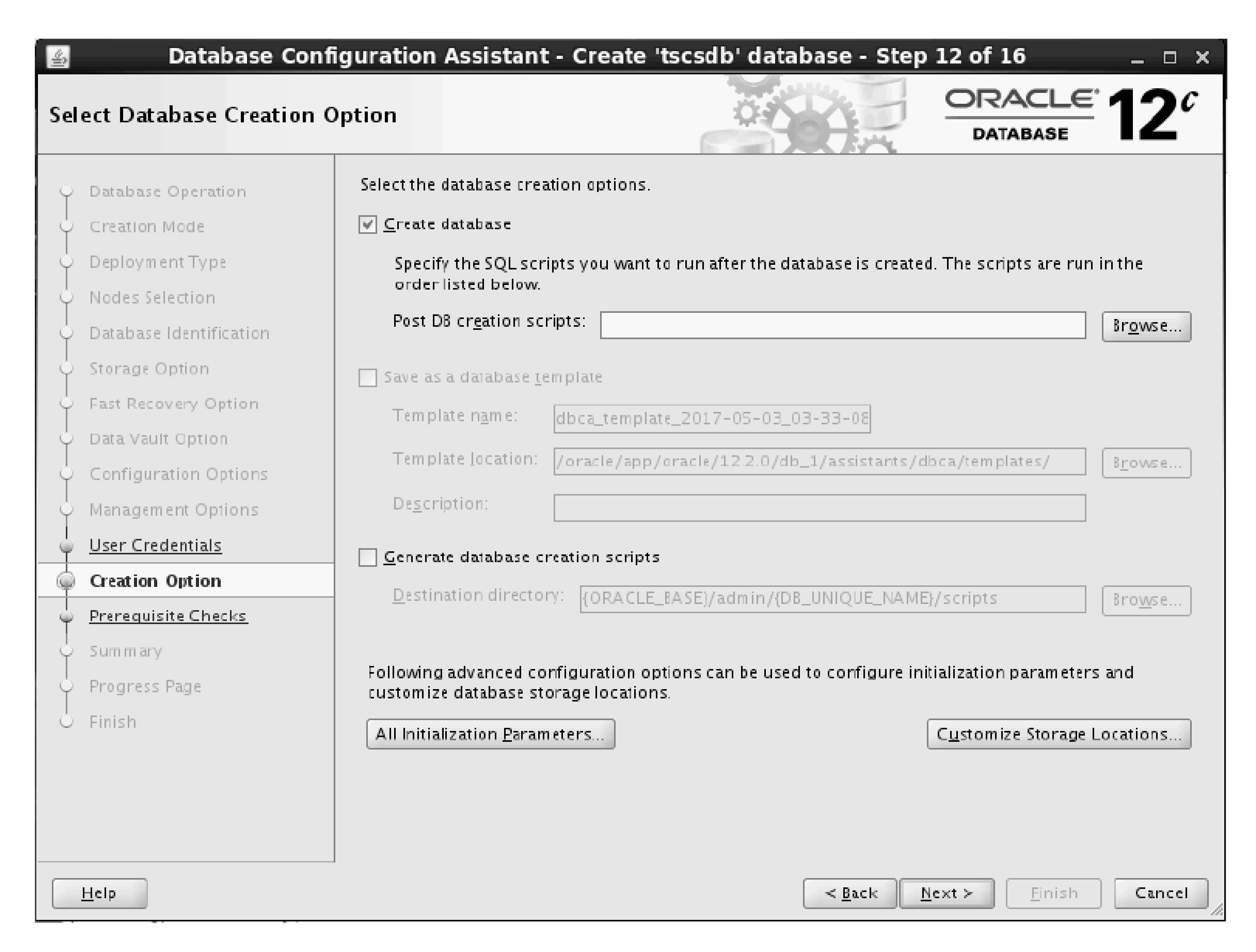


图 12-47　数据库创建选项

14. 执行预检查

如图 12-48 所示，进行安装前的最后检查工作。

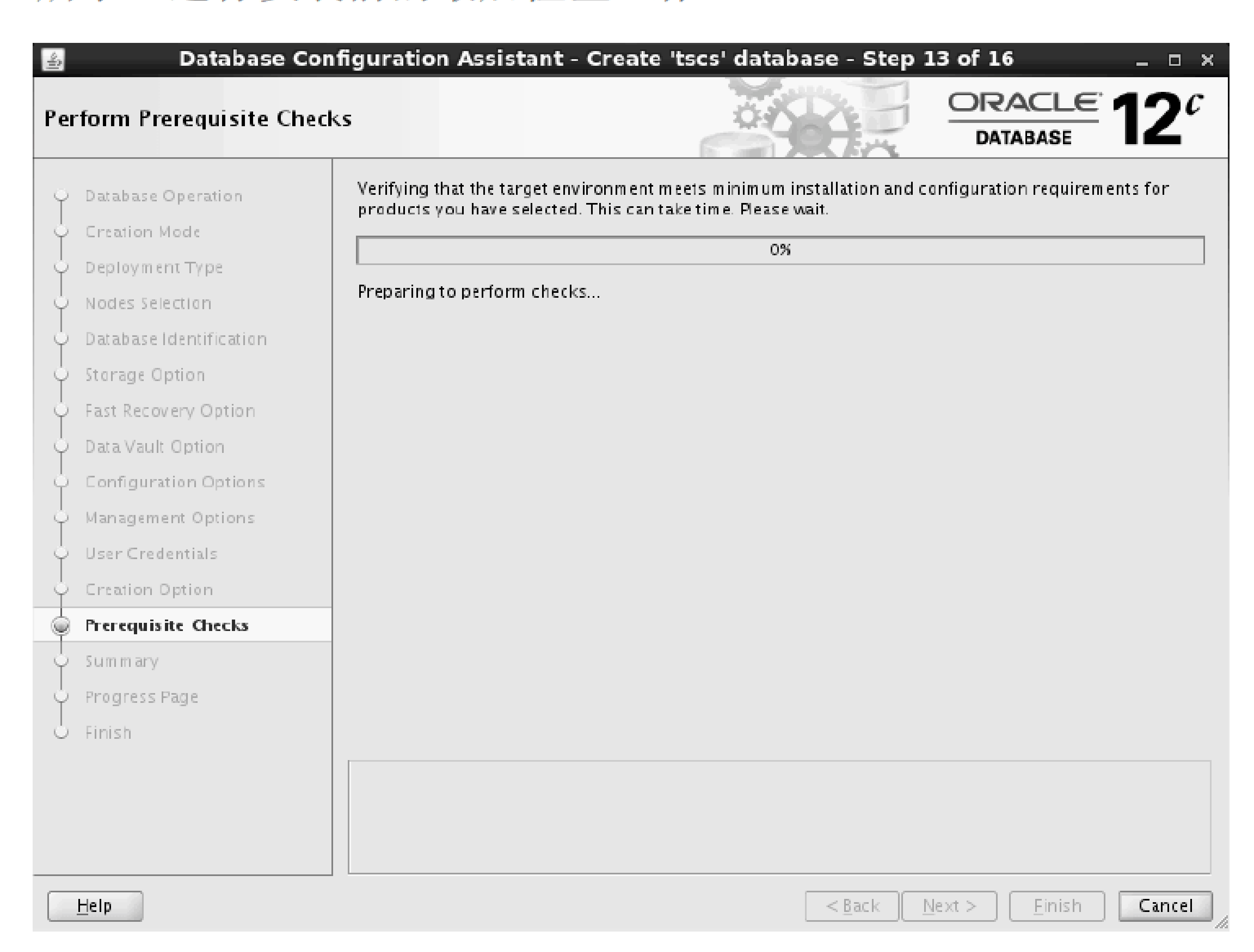


图 12-48　数据库创建预检查

如图 12-49 所示，请勾选右上角的 Ignore ALL 复选框，单击 Next 按钮。

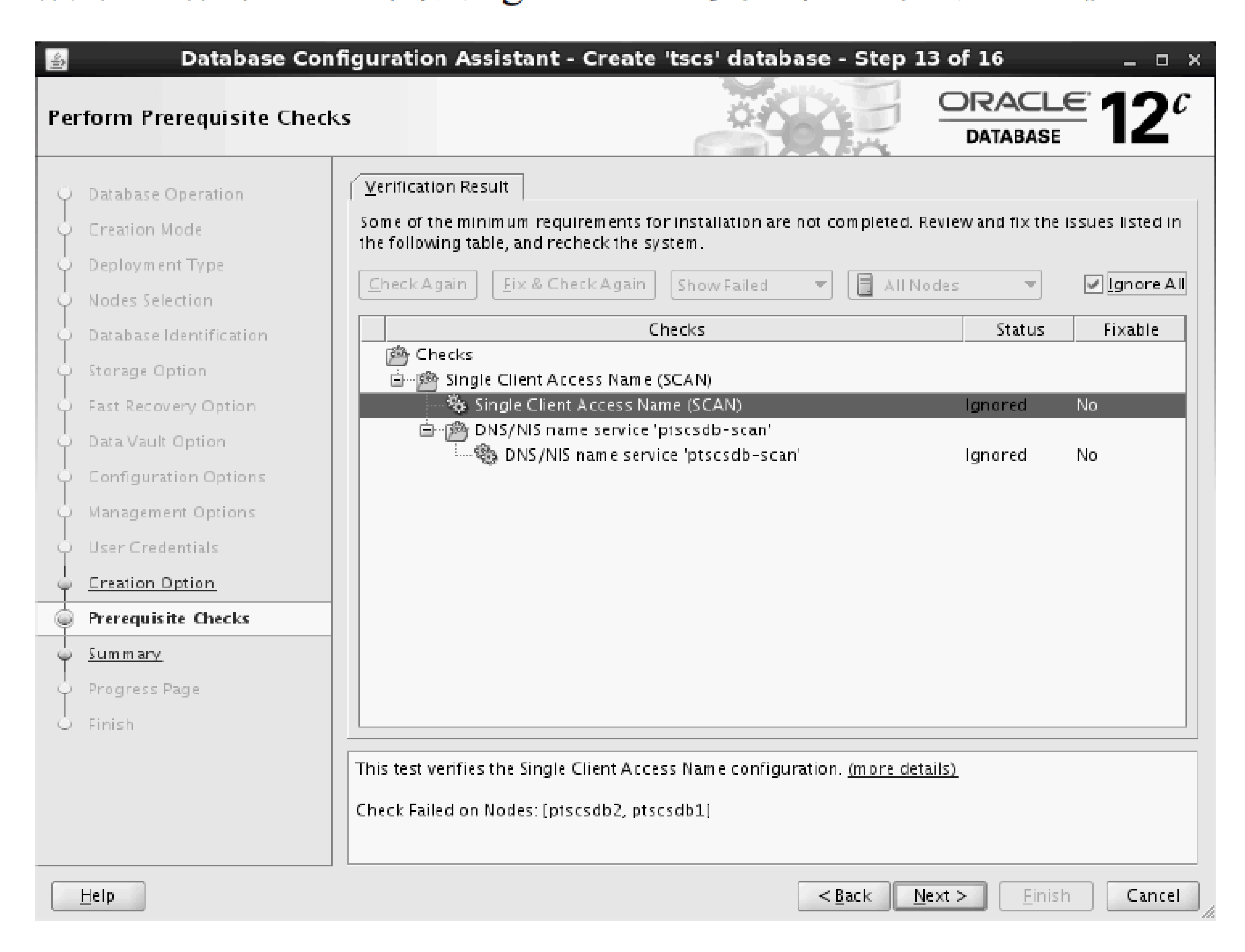


图 12-49　检查结果确认

15. 安装概要

如图 12-50 所示，单击 Finish 按钮开始数据库实例的创建。

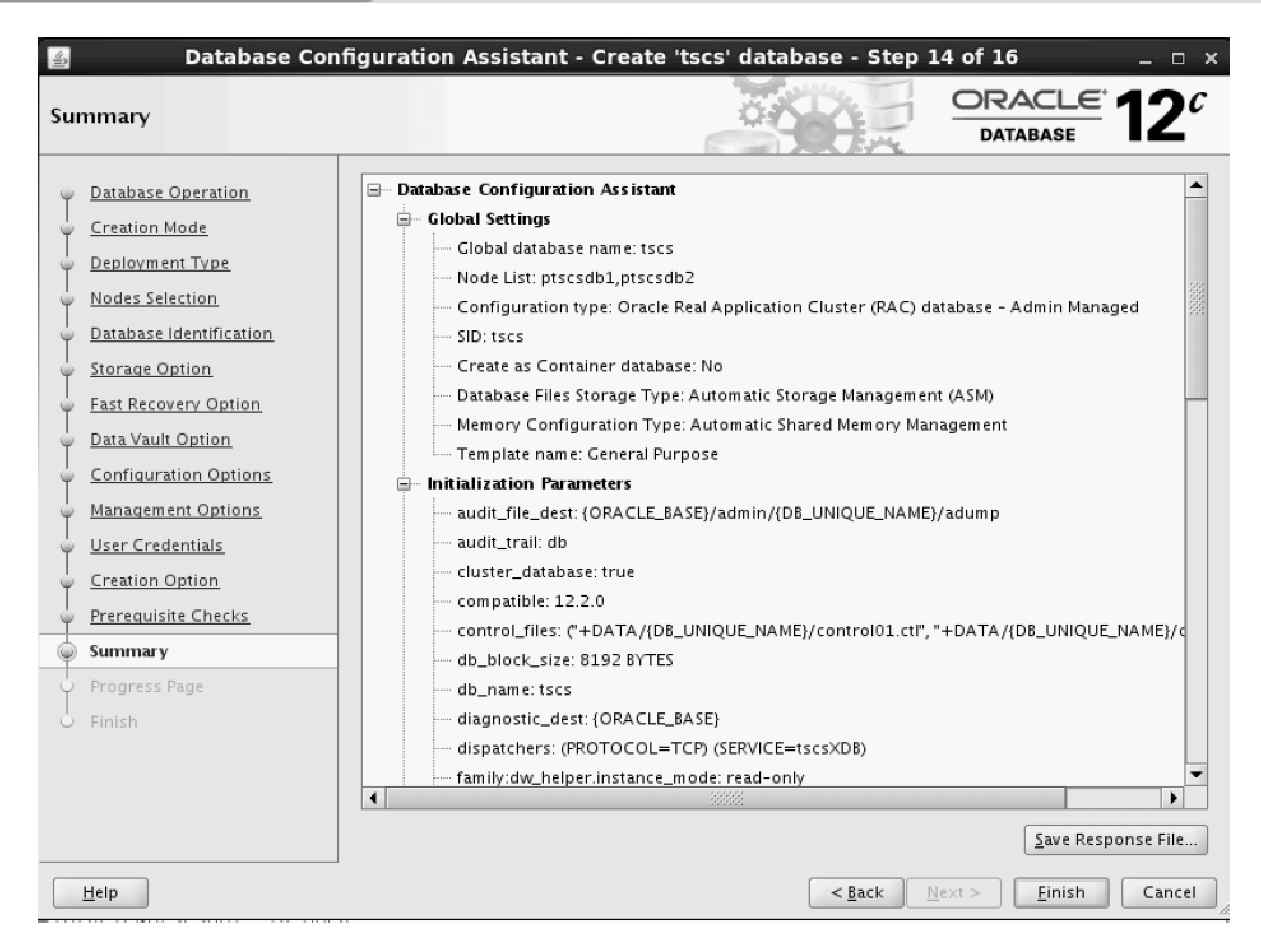


图 12-50 安装概要

如图 12-51 所示，显示数据库实例的创建过程。

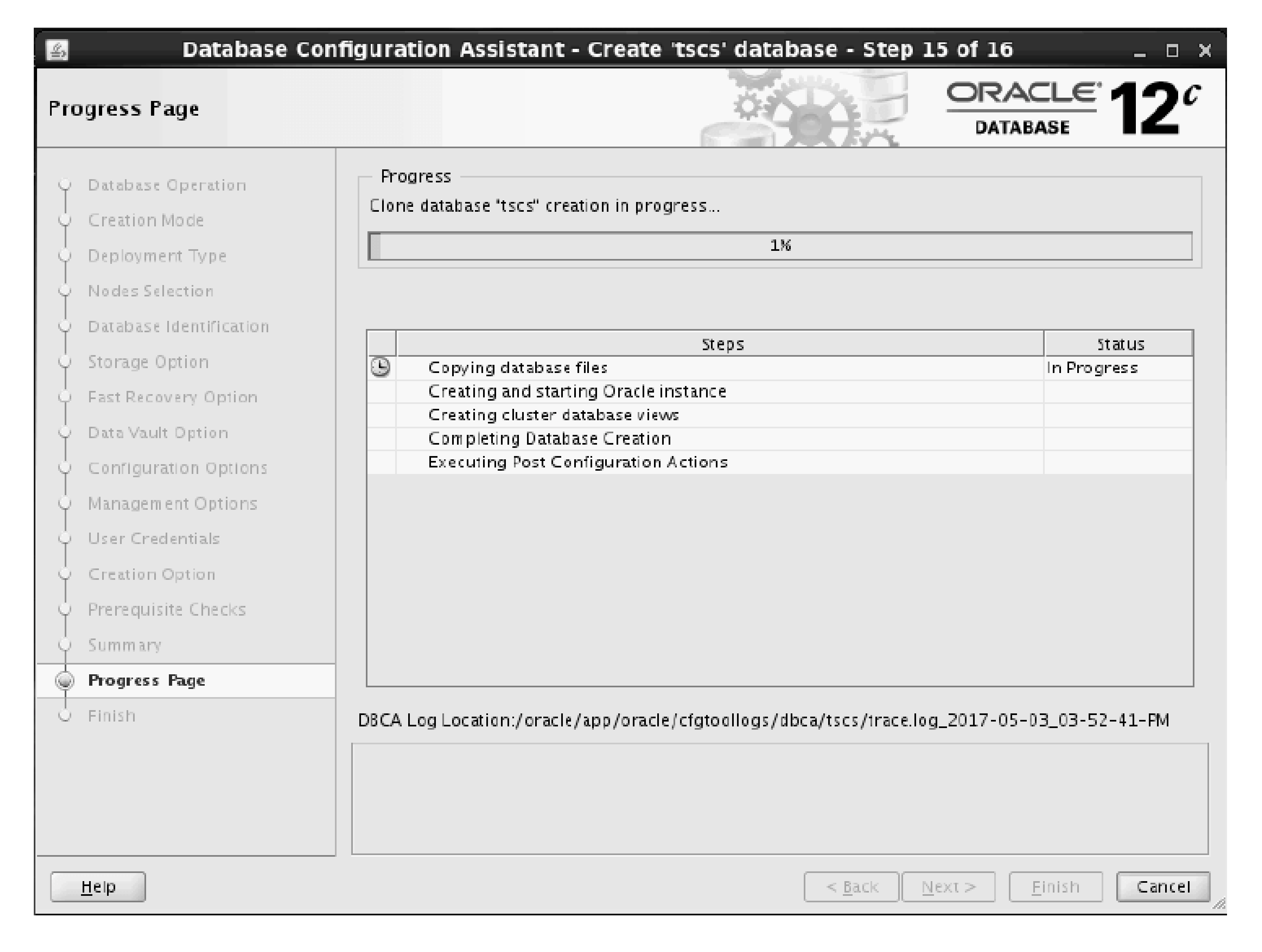


图 12-51 启动安装

16. 安装结束

如图 12-52 所示，数据库实例创建完毕，单击 Close 按钮关闭 DBCA。

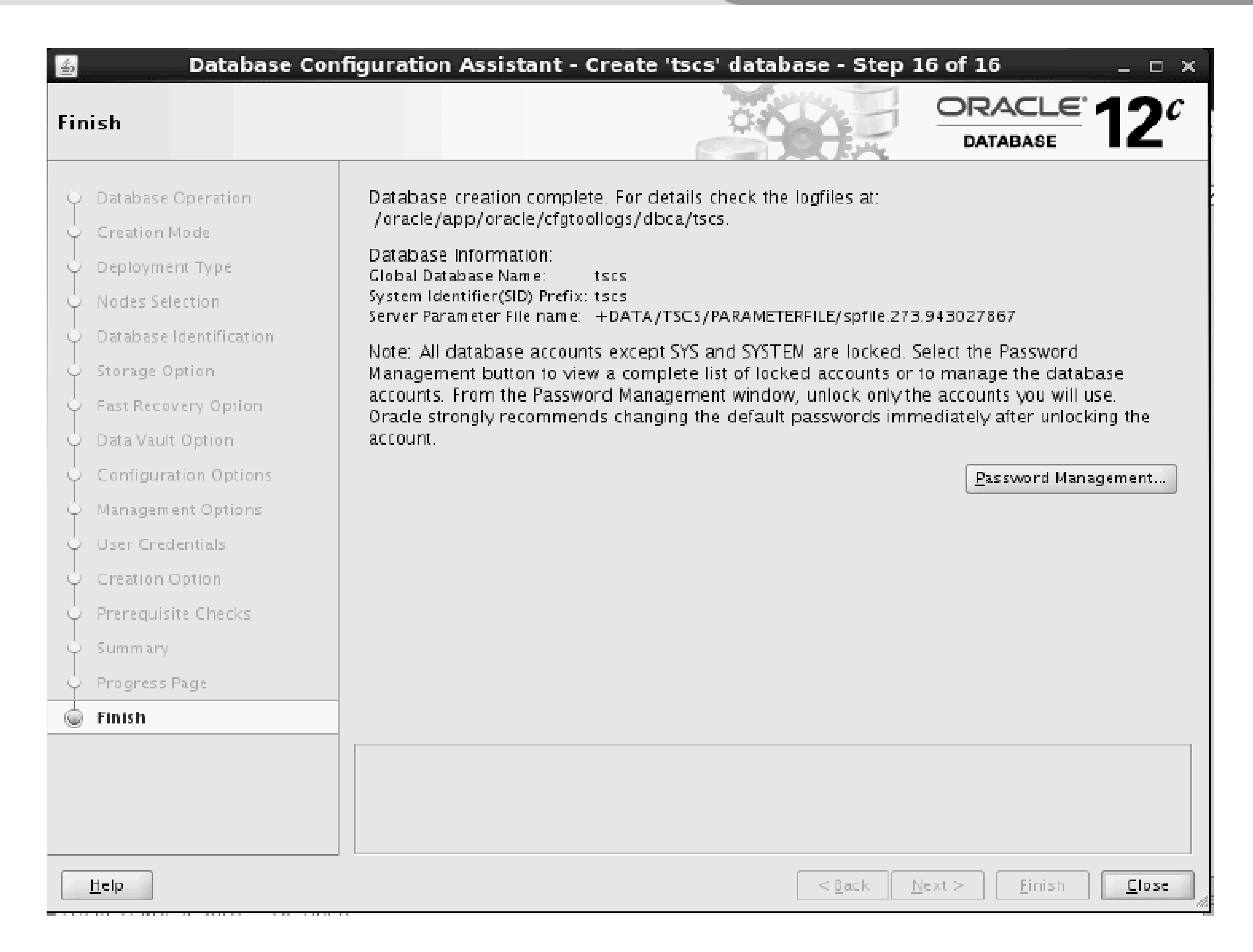


图 12-52　实例创建完毕

12.8 安装 PSU

当前 Oracle 数据库提供两种方式的补丁：

- 主动的 Proactive Patches。
- 被动的 Reactive Patches。

其中 Reactive Patches 是指过去的 ONE-OFF Patch，而过去的 PSU、SPU/CPU、BP 都是 Proactive Patches。从 12c（12.1.0.2）起数据库又提供了一个名为 DBBP 的补丁类型，在数据库安装选择补丁时建议是 PSU、CPU、DBBP 中的一种，在 12c 以前至少是安装相应版本的 PSU，而从 12.1.0.2 起想要安装更全的补丁集应该选择 DBBP。DBBP 更是一种趋势，以后可能会替代 PSU，就像以前的 CPU 后改名为 SPU，安装 SPU 建议与安装 PSU 的方式一样，DBBP 成了更大的超集。下面来整理一份数据库相关的补丁术语及注意事项。

1. patch number 改变

从 2015 年 11 月开始数据库相关的 PSU、SPU、BP 的版本号第 5 位发生改变，如 11204 的 PSU 的 11.2.0.4.8 下一个版本不再是 11.2.0.4.9，而是 11.2.0.4.160119， 格式是与发布日期相关的 YYMMDD，年份的后两位是月份和日期，方便查找 PATCH 的时间段。

2. SPU/CPU 改变

从 12.1.0.1 开始数据库提供安全相关的修复不再单独以 SPU 的形式发布，而是以 PSU 或 DBBP 打包的方式集中修复。这种方式更为简单，DBBP 的内容包含了 PSU 和 SPV。

3. BP FOR EXADATA AND DBIM 改变

从 2016 年 4 月 Database Patch for Engineered Systems and Database In-Memory Bundle Patch（BP）改名为 Database Proactive Bundle Patch，也就是 DBBP。不再仅限于 EXADATA 系统，Database Proactive Bundle Patch 是多个 PSU 的超集，包含了 GI PSU、DB PSU 及 EXPDATA 和 DBIM 的相关修复，从 dba_registry_sqlpatch 视图中可以看到 DBBP 区别于原来的 PSU，也就是从 12.1.0.2 以后 Database Proactive Bundle Patch 可以应用于所有数据库环境。另外，DBBP 仅用于 Linux 和 UNIX 环境，与 Windows 的 BP 没有关系，Windows 平台还继续使用 Bundle Patch（Windows 32bit & 64bit）。

下面将演示 12c 版本下的 PSU 安装过程。

12.8.1 解压、授权

Oracle 的 PSU 安装文件是以压缩包的形式下载下来的，在使用之前需要手动进行解压，并且将解压出来的文件授权给使用者，也就是对应的安装用户。

```
###psu 171017
cd /u01/app/12.2.0.1/grid
mv OPatch Opatchbak
cp -R /u01/install/OPatch .
chown -R grid:oinstall OPatch
cd /u01/app/oracle/product/12.2.0.1/dbhome_1
mv OPatch Opatchbak
cp -R /u01/install/OPatch .
chown -R oracle:oinstall OPatch
```

12.8.2 命令

下面给出具体执行应用补丁的方法，从命令的内容不难看出，PSU 的应用方法跟补丁编号和程序的安装目录有关。这里只给出示例，在实际的实施工作中，还需要读者根据实际情况进行修改，但大体上命令的结构是不变的。

```
su -
opatchauto apply /u01/install/26737266  -oh /u01/app/12.2.0.1/grid
source /home/oracle/.bash_profile
opatchauto apply /u01/install/26737266  -oh
/u01/app/oracle/product/12.2.0.1/dbhome_1
opatchauto rollback /u01/install/25078431  -oh /u01/app/12.2.0.1/grid
```

12.8.3 过程输出

下面给出升级的具体示例，从 RAC 的第 1 节点开始为 grid 程序打 PSU 补丁：

```
[root@zj-vsan-rac14 ~]# opatchauto apply /u01/install/26737266  -oh
/u01/app/12.2.0.1/grid

OPatchauto session is initiated at Fri Jan  5 16:34:07 2018
```

```
System initialization log file is
/u01/app/12.2.0.1/grid/cfgtoollogs/opatchautodb/systemconfig2018-01-05 04-34-1
0PM.log.

Session log file is
/u01/app/12.2.0.1/grid/cfgtoollogs/opatchauto/opatchauto2018-01-05_04-34-20PM.
log
The id for this session is MZJK

Executing OPatch prereq operations to verify patch applicability on home
/u01/app/12.2.0.1/grid
Patch applicability verified successfully on home /u01/app/12.2.0.1/grid


Bringing down CRS service on home /u01/app/12.2.0.1/grid
Prepatch operation log file location:
/u01/app/grid/crsdata/zj-vsan-rac14/crsconfig/crspatch_zj-vsan-rac14_2018-01-0
5_04-35-10PM.log
CRS service brought down successfully on home /u01/app/12.2.0.1/grid


Start applying binary patch on home /u01/app/12.2.0.1/grid

Binary patch applied successfully on home /u01/app/12.2.0.1/grid


Starting CRS service on home /u01/app/12.2.0.1/grid

Postpatch operation log file location:
/u01/app/grid/crsdata/zj-vsan-rac14/crsconfig/crspatch_zj-vsan-rac14_2018-01-0
5 04-51-33PM.log
CRS service started successfully on home /u01/app/12.2.0.1/grid

OPatchAuto successful.

--------------------------------Summary--------------------------------

Patching is completed successfully. Please find the summary as follows:

Host:zj-vsan-rac14
CRS Home:/u01/app/12.2.0.1/grid
Summary:

==Following patches were SUCCESSFULLY applied:

Patch: /u01/install/26737266/26710464
Log:
/u01/app/12.2.0.1/grid/cfgtoollogs/opatchauto/core/opatch/opatch2018-01-05_16-
36-22PM 1.log
```

```
Patch: /u01/install/26737266/26737232
Log:
/u01/app/12.2.0.1/grid/cfgtoollogs/opatchauto/core/opatch/opatch2018-01-05_16-
36-22PM_1.log

Patch: /u01/install/26737266/26839277
Log:
/u01/app/12.2.0.1/grid/cfgtoollogs/opatchauto/core/opatch/opatch2018-01-05_16-
36-22PM_1.log

Patch: /u01/install/26737266/26925644
Log:
/u01/app/12.2.0.1/grid/cfgtoollogs/opatchauto/core/opatch/opatch2018-01-05_16-
36-22PM_1.log

Patch: /u01/install/26737266/26928563
Log:
/u01/app/12.2.0.1/grid/cfgtoollogs/opatchauto/core/opatch/opatch2018-01-05_16-
36-22PM_1.log



OPatchauto session completed at Fri Jan  5 16:57:10 2018
Time taken to complete the session 23 minutes, 3 seconds
```

在 RAC 节点 1 顺利完成 PSU 补丁升级之后，按照顺序开始执行第 2 节点 grid 的 PSU 升级过程：

```
[root@zj-vsan-rac15 install]#  opatchauto apply /u01/install/26737266  -oh
/u01/app/12.2.0.1/grid

OPatchauto session is initiated at Fri Jan  5 17:02:36 2018

System initialization log file is
/u01/app/12.2.0.1/grid/cfgtoollogs/opatchautodb/systemconfig2018-01-05_05-02-3
9PM.log.

Session log file is
/u01/app/12.2.0.1/grid/cfgtoollogs/opatchauto/opatchauto2018-01-05_05-02-59PM.
log
The id for this session is EF41

Executing OPatch prereq operations to verify patch applicability on home
/u01/app/12.2.0.1/grid
Patch applicability verified successfully on home /u01/app/12.2.0.1/grid


Bringing down CRS service on home /u01/app/12.2.0.1/grid
```

```
Prepatch operation log file location:
/u01/app/grid/crsdata/zj-vsan-rac15/crsconfig/crspatch_zj-vsan-rac15_2018-01-0
5 05-03-50PM.log
CRS service brought down successfully on home /u01/app/12.2.0.1/grid


Start applying binary patch on home /u01/app/12.2.0.1/grid




Binary patch applied successfully on home /u01/app/12.2.0.1/grid


Starting CRS service on home /u01/app/12.2.0.1/grid




Postpatch operation log file location:
/u01/app/grid/crsdata/zj-vsan-rac15/crsconfig/crspatch zj-vsan-rac15 2018-01-0
5_05-20-01PM.log
CRS service started successfully on home /u01/app/12.2.0.1/grid

OPatchAuto successful.

--------------------------------Summary--------------------------------

Patching is completed successfully. Please find the summary as follows:

Host:zj-vsan-rac15
CRS Home:/u01/app/12.2.0.1/grid
Summary:

==Following patches were SUCCESSFULLY applied:

Patch: /u01/install/26737266/26710464
Log:
/u01/app/12.2.0.1/grid/cfgtoollogs/opatchauto/core/opatch/opatch2018-01-05 17-
05-13PM_1.log

Patch: /u01/install/26737266/26737232
Log:
/u01/app/12.2.0.1/grid/cfgtoollogs/opatchauto/core/opatch/opatch2018-01-05 17-
05-13PM 1.log

Patch: /u01/install/26737266/26839277
Log:
/u01/app/12.2.0.1/grid/cfgtoollogs/opatchauto/core/opatch/opatch2018-01-05_17-
05-13PM 1.log
```

```
Patch: /u01/install/26737266/26925644
Log:
/u01/app/12.2.0.1/grid/cfgtoollogs/opatchauto/core/opatch/opatch2018-01-05 17-
05-13PM 1.log

Patch: /u01/install/26737266/26928563
Log:
/u01/app/12.2.0.1/grid/cfgtoollogs/opatchauto/core/opatch/opatch2018-01-05 17-
05-13PM 1.log



OPatchauto session completed at Fri Jan  5 17:22:20 2018
Time taken to complete the session 19 minutes, 44 seconds
```

针对 Oracle RAC，应用程序应该包含两部分，首先是基础的部分，即集群软件，之后是数据库软件。从 RAC 的安装过程也可以看出，集群软件是基础环境，数据库软件是建立在集群软件之上的应用。所以，PSU 的升级过程也同样要求先对 grid 集群软件实施，集群升级完毕之后，再进行数据库软件的 PSU 升级。下面开始针对 RAC 第 1 节点的数据库软件进行 PSU 升级，过程如下：

```
[root@zj-vsan-rac14 ~]#
/u01/app/oracle/product/12.2.0.1/dbhome 1/OPatch/opatchauto apply
/u01/install/26737266  -oh /u01/app/oracle/product/12.2.0.1/dbhome_1

OPatchauto session is initiated at Fri Jan  5 17:22:12 2018

System initialization log file is
/u01/app/oracle/product/12.2.0.1/dbhome 1/cfgtoollogs/opatchautodb/systemconfi
g2018-01-05_05-22-15PM.log.

Session log file is
/u01/app/oracle/product/12.2.0.1/dbhome_1/cfgtoollogs/opatchauto/opatchauto201
8-01-05 05-22-44PM.log
The id for this session is 82LQ

Executing OPatch prereq operations to verify patch applicability on home
/u01/app/oracle/product/12.2.0.1/dbhome_1
Patch applicability verified successfully on home
/u01/app/oracle/product/12.2.0.1/dbhome 1


Verifying SQL patch applicability on home
/u01/app/oracle/product/12.2.0.1/dbhome_1
No step execution required.........
SQL patch applicability verified successfully on home
```

```
/u01/app/oracle/product/12.2.0.1/dbhome_1


Preparing to bring down database service on home
/u01/app/oracle/product/12.2.0.1/dbhome_1
No step execution required.........
Successfully prepared home /u01/app/oracle/product/12.2.0.1/dbhome_1 to bring down
database service


Performing prepatch operation on home /u01/app/oracle/product/12.2.0.1/dbhome_1
Perpatch operation completed successfully on home
/u01/app/oracle/product/12.2.0.1/dbhome_1


Start applying binary patch on home /u01/app/oracle/product/12.2.0.1/dbhome_1



 Binary patch applied successfully on home
/u01/app/oracle/product/12.2.0.1/dbhome 1


Performing postpatch operation on home /u01/app/oracle/product/12.2.0.1/dbhome_1
Postpatch operation completed successfully on home
/u01/app/oracle/product/12.2.0.1/dbhome 1


Preparing home /u01/app/oracle/product/12.2.0.1/dbhome_1 after database service
restarted
No step execution required.........
Prepared home /u01/app/oracle/product/12.2.0.1/dbhome 1 successfully after
database service restarted


Trying to apply SQL patch on home /u01/app/oracle/product/12.2.0.1/dbhome_1
No step execution required.........
SQL patch applied successfully on home /u01/app/oracle/product/12.2.0.1/dbhome 1

OPatchAuto successful.

--------------------------------Summary--------------------------------

Patching is completed successfully. Please find the summary as follows:

Host:zj-vsan-rac14
RAC Home:/u01/app/oracle/product/12.2.0.1/dbhome_1
Summary:

==Following patches were SKIPPED:
```

```
Patch: /u01/install/26737266/26737232
Reason: This patch is not applicable to this specified target type - "rac_database"

Patch: /u01/install/26737266/26839277
Reason: This patch is not applicable to this specified target type - "rac_database"

Patch: /u01/install/26737266/26928563
Reason: This patch is not applicable to this specified target type - "rac_database"


==Following patches were SUCCESSFULLY applied:

Patch: /u01/install/26737266/26710464
Log:
/u01/app/oracle/product/12.2.0.1/dbhome_1/cfgtoollogs/opatchauto/core/opatch/o
patch2018-01-05_17-22-59PM_1.log

Patch: /u01/install/26737266/26925644
Log:
/u01/app/oracle/product/12.2.0.1/dbhome_1/cfgtoollogs/opatchauto/core/opatch/o
patch2018-01-05_17-22-59PM_1.log



OPatchauto session completed at Fri Jan  5 17:35:34 2018
Time taken to complete the session 13 minutes, 22 seconds
```

第 1 节点升级完毕之后，下面开始对第 2 节点进行 PSU 的升级，过程如下：

```
[root@zj-vsan-rac15 ~]#
/u01/app/oracle/product/12.2.0.1/dbhome_1/OPatch/opatchauto apply
/u01/install/26737266  -oh /u01/app/oracle/product/12.2.0.1/dbhome_1

OPatchauto session is initiated at Fri Jan  5 17:39:33 2018

System initialization log file is
/u01/app/oracle/product/12.2.0.1/dbhome_1/cfgtoollogs/opatchautodb/systemconfi
g2018-01-05_05-39-36PM.log.

Session log file is
/u01/app/oracle/product/12.2.0.1/dbhome_1/cfgtoollogs/opatchauto/opatchauto201
8-01-05_05-39-54PM.log
The id for this session is 5ML4

Executing OPatch prereq operations to verify patch applicability on home
/u01/app/oracle/product/12.2.0.1/dbhome_1
Patch applicability verified successfully on home
/u01/app/oracle/product/12.2.0.1/dbhome_1


Verifying SQL patch applicability on home
```

```
/u01/app/oracle/product/12.2.0.1/dbhome_1
No step execution required.........
SQL patch applicability verified successfully on home
/u01/app/oracle/product/12.2.0.1/dbhome 1


Preparing to bring down database service on home
/u01/app/oracle/product/12.2.0.1/dbhome 1
No step execution required.........
Successfully prepared home /u01/app/oracle/product/12.2.0.1/dbhome_1 to bring down
database service


Performing prepatch operation on home /u01/app/oracle/product/12.2.0.1/dbhome 1
Perpatch operation completed successfully on home
/u01/app/oracle/product/12.2.0.1/dbhome_1


Start applying binary patch on home /u01/app/oracle/product/12.2.0.1/dbhome_1




Binary patch applied successfully on home
/u01/app/oracle/product/12.2.0.1/dbhome_1


Performing postpatch operation on home /u01/app/oracle/product/12.2.0.1/dbhome_1
Postpatch operation completed successfully on home
/u01/app/oracle/product/12.2.0.1/dbhome_1


Preparing home /u01/app/oracle/product/12.2.0.1/dbhome 1 after database service
restarted
No step execution required.........
Prepared home /u01/app/oracle/product/12.2.0.1/dbhome_1 successfully after
database service restarted


Trying to apply SQL patch on home /u01/app/oracle/product/12.2.0.1/dbhome_1
No step execution required.........
SQL patch applied successfully on home /u01/app/oracle/product/12.2.0.1/dbhome_1

OPatchAuto successful.

--------------------------------Summary--------------------------------

Patching is completed successfully. Please find the summary as follows:

Host:zj-vsan-rac15
RAC Home:/u01/app/oracle/product/12.2.0.1/dbhome_1
```

```
Summary:

==Following patches were SKIPPED:

Patch: /u01/install/26737266/26737232
Reason: This patch is not applicable to this specified target type - "rac_database"

Patch: /u01/install/26737266/26839277
Reason: This patch is not applicable to this specified target type - "rac database"

Patch: /u01/install/26737266/26928563
Reason: This patch is not applicable to this specified target type - "rac_database"


==Following patches were SUCCESSFULLY applied:

Patch: /u01/install/26737266/26710464
Log:
/u01/app/oracle/product/12.2.0.1/dbhome_1/cfgtoollogs/opatchauto/core/opatch/o
patch2018-01-05 17-40-10PM 1.log

Patch: /u01/install/26737266/26925644
Log:
/u01/app/oracle/product/12.2.0.1/dbhome_1/cfgtoollogs/opatchauto/core/opatch/o
patch2018-01-05_17-40-10PM_1.log


OPatchauto session completed at Fri Jan  5 17:53:00 2018
Time taken to complete the session 13 minutes, 27 seconds
```

请读者注意观察升级过程的程序文本输出，顺利的情况下不应有报错产生。

12.9 卸载

关于软件的卸载，Oracle 提供了专门的卸载脚本，如有卸载需求，只需执行该脚本即可。

12.9.1 卸载 Database Software

Oracle 的数据库软件卸载脚本文件所在的位置如下：

```
$ORACLE_HOME/deinstall/deinstall
```

在执行卸载脚本的时候，需要注意以下三个可选项。

- home：检查 oracle home 的位置。
- -silent：使用静默或者响应文件模式卸载，软件会自动找到响应文件，默认响应文件的位置是在 ORACLE_HOME/deinstall/response 目录下的 deinstall.rsp.tmpl。

- -checkonly：仅仅检查已安装 Oracle 的配置。

12.9.2 卸载 Grid Infrastructure

Oracle 的集群软件卸载脚本文件所在的位置如下。下面给出命令的结构和可选项，具体使用的时候，可以作为参考。

```
$ORACLE_HOME/deinstall/deinstall
deinstall -home <Complete path of Oracle home>
             [ -silent ]
             [ -checkonly ]
             [ -local ]
             [ -paramfile <complete path of input parameter property file> ]
             [ -params <name1=value[ name2=value ...]> ]
             [ -o <complete path of directory for saving files> ]
             [ -help | -h: Type -h or -help to get more information on each o
f the above options. ]
Specify any of the above options.
```

从日常运维的角度来看，Oracle 软件无论是数据库还是集群，卸载的情况比较少见，也不作为重点的学习内容，这里简单向读者介绍命令的位置和使用方法。有兴趣的读者可以在业余时间自行实验。

12.10 小结

本章介绍了 Oracle RAC 的安装部署方法。一般来讲，RAC 的安装属于 Oracle 软件的高级操作，该架构在大型企业中运用较多，因为其对系统数据库可靠性有更高的要求，如果读者可以熟练掌握 RAC 安装方法，说明 Oracle 数据库的学习已经有了一个良好的开端。